SHELLBOXES

Leveraged Looping Strategies

Smart Contract Security Audit

Prepared by ShellBoxes

July 30th, 2025 - August 12th, 2025

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Raga Finance
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
`https://github.com/Nexus-2023/NativeYield`	86511a1600ecf8751b7d2007928e87789598ed21

Re-Audit

Repository	Commit Hash
`https://github.com/Nexus-2023/NativeYield`	3749b2bd969ad9564f685cb26ddaa97ea7c86fd8

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Raga Finance engaged ShellBoxes to conduct a security assessment on the Leveraged Looping Strategies beginning on July 30th, 2025 and ending August 12th, 2025. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Raga Finance

Raga Finance brings institutional-grade structured products to help DeFi users maximize returns. Their one-click vaults simplify access to advanced yield strategies like delta neutral, autocompounding, yield looping, and more

Issuer	Raga Finance
Website	`https://www.raga.finance`
Type	Solidity Smart Contract
Documentation	Raga Finance Docs
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact

High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Leveraged Looping Strategies implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 3 high-severity, 5 medium-severity, 10 low-severity, 4 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Inverted Leverage Bounds Allow Invalid Leverage	HIGH	Fixed
SHB.2. Missing Slippage Controls In Strategy Swaps	HIGH	Fixed
SHB.3. Unsafe And Sticky ERC-20 Allowance/Transfer Patterns	HIGH	Fixed
SHB.4. Delegatecall Into Strategy Logic Can Mutate Vault Storage	MEDIUM	Fixed
SHB.5. Division-By-Zero Across Leverage And Withdrawal Math	MEDIUM	Fixed
SHB.6. Oracle Price Freshness Unchecked	MEDIUM	Fixed

SHB.7. Phishing Vault Creation Via Permissionless createVault	MEDIUM	Fixed
SHB.8. UserVault Lacks Reentrancy Protection	MEDIUM	Fixed
SHB.9. Centralized Upgrade And Strategy Whitelisting	LOW	Acknowledged
SHB.10. Missing Explicit Strategy Address Check Before Delegatecall	LOW	Fixed
SHB.11. Missing Lending Threshold Cap	LOW	Fixed
SHB.12. Missing Zero Address Validation	LOW	Fixed
SHB.13. Non-Terminating Withdraw Loop	LOW	Fixed
SHB.14. Price Feed Conversions Lack Sanity Bounds	LOW	Acknowledged
SHB.15. Pull-Before-Check Deposit Flow	LOW	Fixed
SHB.16. Strategy Validation Missing	LOW	Fixed
SHB.17. Vault Ownership Centralized In Registry	LOW	Acknowledged
SHB.18. VaultRegistry May Operate With Unset vaultManager	LOW	Fixed
SHB.19. Decimals Misconfiguration Skews Math	INFORMATIONAL	Acknowledged
SHB.20. Misnamed Access Modifier And Brittle Auth Coupling	INFORMATIONAL	Fixed
SHB.21. Payable Functions Ignore Msg.Value	INFORMATIONAL	Fixed
SHB.22. Underflow Risk In Full-Withdraw Preview	INFORMATIONAL	Fixed

3 Finding Details

SHB.1 Inverted Leverage Bounds Allow Invalid Leverage

- Severity : HIGH
- Status : Fixed

- Likelihood : 2
- Impact : 3

Description:

The leverage guard uses an impossible condition, allowing leverage outside the intended range to bypass validation. Both read and write paths rely on `leverageInWAD < WAD && leverageInWAD > _getMaxLeverage()`, effectively disabling min/max leverage enforcement.

Files Affected:

SHB.1.1: Leverage Check

```
if (leverageInWAD < WAD && leverageInWAD > _getMaxLeverage()) revert;
```

Recommendation:

Replace the `&&` with | and revert when leverage is below 1× or above the maximum. Add boundary tests to prevent future regression.

Updates

The team resolved the issue by changing the and operator to an or operator.

SHB.2 Missing Slippage Controls In Strategy Swaps

- Severity : HIGH
- Status : Fixed

- Likelihood : 2
- Impact : 3

Description:

Internal swaps such as `_swapBorrowToCollateral` and `_swapPTForIBT` pass `minOut = 0`, exposing users to MEV and sandwich attacks during large deposits or withdrawals. Without caller-specified slippage limits, an attacker can drain value via unfavorable rates.

Files Affected:

SHB.2.1: Swap Helper

```
_swapBorrowToCollateral(amountIn, 0);
```

Recommendation:

Accept user-supplied `minOut`/`maxSlippage` parameters and enforce them in every swap to bound price impact.

Updates

The team resolved the issue by adding the minShares variable that act as a slippage protection.

SHB.3 Unsafe And Sticky ERC-20 Allowance/Transfer Patterns

- Severity: HIGH
- Status: Fixed

- Likelihood: 2
- Impact: 3

Description:

Multiple modules approve or transfer tokens directly without zeroing allowances or using SafeERC20 helpers, leaving residual spend rights and risking silent failures with non-standard tokens. Examples include PositionManager._deposit, BaseSpectraUtils swap helpers, flash-loan callbacks, and Ajna/Morpho debt functions.

Files Affected:

SHB.3.1: PositionManager.sol

```
IERC20(_token).approve(vault, _value);
```

SHB.3.2: AjnaBorrowLendingStrategy.sol

```
borrow_token.approve(address(erc20Pool), debtToRepay);
// ... external call ...
```

Recommendation:

Use SafeERC20 wrappers and the zero-first approval pattern, approving exact amounts and resetting allowances to zero after use.

Updates

The team resolved the issue by using the forceApprove function in the SafeERC20 library.

SHB.4 Delegatecall Into Strategy Logic Can Mutate Vault Storage

- Severity : MEDIUM
- Status : Fixed

- Likelihood : 2
- Impact : 2

Description:

UserVault delegates deposits, withdrawals, and flash-loan callbacks to strategy contracts, executing in the vault's storage context. A bug or malicious upgrade in a whitelisted strategy can corrupt vault state or drain funds if governance is compromised.

Files Affected:

SHB.4.1: UserVault.sol

```
Address.functionDelegateCall(address(_strategy), data);
```

Recommendation:

Treat strategies as trusted code: restrict upgrades via multisig + timelock, emit strategy change events, and consider safer `call`-based patterns where feasible.

Updates

The team resolved the issue by adding an onlyValidStrategy modifier. In case a strategy is malicious or has unintended behavior, it can be removed from the strategyManager. They also implemented pause functions at both the strategy and vault levels. Pausing a strategy prevents users from depositing additional funds into it.

SHB.5 Division-By-Zero Across Leverage And Withdrawal Math

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

Functions such as `_calculateWithdrawalAmountByShares`, `_getLeverage`, and `getCurrentStageEffectiveAmount` divide by user-supplied values like `_shares`, `collateralWeight`, or `(WAD - adjustedRiskRatio)` without guarding zero or boundary cases. Zero denominators revert unexpectedly, enabling denial-of-service on deposits and withdrawals.

Recommendation:

Validate all denominators and risk ratios before division, reverting with clear errors when zero or out-of-range.

Updates

The team resolved the issue by adding the necessary validations.

SHB.6 Oracle Price Freshness Unchecked

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

`BOLDPriceFeed` calls Pyth's `getPriceUnsafe` and only checks `price.price > 0`, ignoring publish time, staleness, and confidence intervals. Stale or low-confidence prices can corrupt USD conversions.

Recommendation:

Use `getPriceNoOlderThan` or validate `publishTime` and confidence ratios before using prices, reverting on stale or uncertain data.

Updates

The team resolved the issue by adding a check for the publish time of the price.

SHB.7 Phishing Vault Creation Via Permissionless createVault

- Severity : MEDIUM
- Status : Fixed

- Likelihood : 2
- Impact : 2

Description:

`VaultRegistry.createVault` is permissionless, enabling attackers to create vaults that appear owned by victims and map them to malicious strategies. Unsuspecting users can be tricked into depositing into attacker-controlled vaults.

Files Affected:

SHB.7.1: VaultRegistry.sol

```
function createVault(address _forUser, ...) external {
    // permissionless creation
}
```

Recommendation:

Require `msg.sender == _forUser` or restrict vault creation to a trusted manager or governance role.

Updates

The team resolved the issue by adding an onlyPositionManager to prevent a user from calling the function directly.

SHB.8 UserVault Lacks Reentrancy Protection

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

`deposit`, `withdraw`, and flash-loan callbacks perform external calls without `nonReentrant` guards. A malicious strategy or token could re-enter mid-operation and manipulate vault state.

Recommendation:

Inherit `ReentrancyGuardUpgradeable` and apply `nonReentrant` to all state-changing external functions.

Updates

The team resolved the issue by adding a reentrancy guard to the deposit and withdrawal functions. For onFlashLoan, the call is made only as a callback, so they added an onlyMorphoBlueCaller check to ensure this function can only be called by the designated callback callee.

SHB.9 Centralized Upgrade And Strategy Whitelisting

- Severity : LOW
- Status : Acknowledged

- Likelihood : 1
- Impact : 2

Description:

`VaultBeacon`, `StrategyManager`, and `VaultManager` are all controlled by single owners, allowing instant upgrades or strategy additions. A compromised key could swap implementations or whitelist malicious strategies across all vaults.

Recommendation:

Adopt multisig ownership with timelocks for upgrades and whitelisting, and emit events to support monitoring.

Updates

The team acknowledged the issue and stated that they will either add additional roles later on or shift the ownership to a multisig.

SHB.10 Missing Explicit Strategy Address Check Before Delegatecall

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

Delegatecall sites do not verify that `_strategy` is non-zero, so misconfigured vaults revert with an opaque error when strategies are unset. This complicates debugging and may mask configuration mistakes.

Recommendation:

Require `address(_strategy)!= address(0)` before delegatecalls and expose a view getter for the configured strategy.

Updates

The team resolved the issue by adding the necessary validations.

SHB.11 Missing Lending Threshold Cap

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

`registerVault` stores `lendingThreshold` without bounds; setting it to zero allows strategies to borrow unlimited collateral. This undermines risk limits and can lead to under-collateralized positions.

Recommendation:

Validate `lendingThreshold` within an expected range (e.g., `1 <= lendingThreshold <= 1` ↪ `e18`) and emit events on changes.

Updates

The team resolved the issue by adding the necessary validations.

SHB.12 Missing Zero Address Validation

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Initializers and setters in `PositionManager`, `VaultManager`, and `StrategyManager` accept zero addresses for critical roles, enabling misconfiguration that bricks system components.

Recommendation:

Add `require(addr != address(0))` checks for all critical addresses during initialization and updates.

Updates

The team resolved the issue by adding the necessary validations.

SHB.13 Non-Terminating Withdraw Loop

- Severity: LOW
- Status: Fixed
- Likelihood: 1
- Impact: 2

Description:

`_withdrawAll` loops `while (debt > 0 && n < 10)`; rounding errors can leave residual debt and cause the function to revert after ten iterations, trapping user funds.

Files Affected:

SHB.13.1: Withdraw Loop

```
while (debt > 0 && n < 10) {
    ...
    n++;
}
```

Recommendation:

Break when residual debt falls below a minimum threshold or settle remaining debt via flash-repay, emitting events for any leftover amount.

Updates

The team resolved the issue by adding the necessary validations.

SHB.14 Price Feed Conversions Lack Sanity Bounds

- Severity : LOW
- Status : Acknowledged

- Likelihood : 1
- Impact : 2

Description:

`SpectraPTIBTPriceFeed`, `sBOLDPriceFeed`, and `SpectrasBOLDPriceFeed` rely on spot rates without slippage or sanity checks, allowing manipulated pool states or lagging rates to distort USD valuations.

Recommendation:

Introduce min/max caps, TWAP comparisons, or rate-band checks and revert on zero or extreme deviations.

Updates

The team acknowledged the issue and stated that these functions are only called to view a user's balance. They are not treated as the source of truth during fund transfers and slippage checks.

SHB.15 Pull-Before-Check Deposit Flow

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

_deposit transfers tokens via transferFrom before confirming that the vault's strategy accepts the asset. If compatibility checks fail afterward, tokens remain stuck in the manager.

Recommendation:

Validate strategy and token compatibility before performing transferFrom, ensuring funds are only moved when the strategy is known to accept them.

Updates

The team resolved the issue by adding the necessary validations.

SHB.16 Strategy Validation Missing

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

addStrategy stores any address without verifying code.length or a version hash, allowing empty or self-destructed contracts to be registered and later redeployed with malicious logic.

Recommendation:

Require _strategy.code.length > 0 and validate a version hash; gate additions through timelocked multisig governance.

Updates

The team resolved the issue by adding the necessary validations.

SHB.17 Vault Ownership Centralized In Registry

- Severity : LOW
- Status : Acknowledged

- Likelihood : 1
- Impact : 2

Description:

`VaultRegistry` becomes the owner of each `UserVault`, preventing users from managing their own vaults and creating a single point of failure.

Recommendation:

Assign ownership to the end user at deployment or allow ownership transfer after creation.

Updates

The team acknowledged the issue and stated that the UserVault contract is called internally and is not public-facing. It is designed to have single ownership because they do not want individual users to have authorization over owner-level responsibilities such as contract upgrades or mapping changes. These changes should only flow through the VaultRegistry and VaultManager.

SHB.18 VaultRegistry May Operate With Unset vaultManager

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

VaultRegistry.createVault relies on vaultManager but never ensures it is non-zero, leading to reverts or misconfigured vaults if updateVaultManager is skipped.

Recommendation:

Require vaultManager != address(0) before vault creation and initialize it during contract setup.

Updates

The team resolved the issue by adding the necessary validations.

SHB.19 Decimals Misconfiguration Skews Math

- Severity : **INFORMATIONAL**
- Status : Acknowledged
- Likelihood : 0
- Impact : 2

Description:

Constants.USDC_DECIMALS is set to 18 instead of the typical 6, and conversions multiply by hard-coded decimal factors across modules. Incorrect constants or assumptions can silently skew accounting by orders of magnitude.

Files Affected:

SHB.19.1: Constants.sol

```
uint8 constant USDC_DECIMALS = 18;
```

Recommendation:

Read decimals dynamically from tokens, cache them, and assert expected values during initialization.

Updates

The team acknowledged the issue and stated that "Here, USDC_DECIMALS is not used to store or convert balances in USDC tokens. It is used as a precision value to convert other token balances into their dollar value". That being said, we have changed the variable name from USDC_DECIMALS to DOLLAR_BALANCE_PRECISION in order to avoid any future confusion."

SHB.20 Misnamed Access Modifier And Brittle Auth Coupling

- Severity : INFORMATIONAL
- Status : Fixed
- Likelihood : 0
- Impact : 1

Description:

The modifier `onlyPositionManger` is misspelled and rigidly enforces `msg.sender == _getPositionManager()`. Misconfiguration blocks all deposits and withdrawals and the typo hinders reviews.

Files Affected:

SHB.20.1: Access Modifier

```
modifier onlyPositionManger() {
    require(msg.sender == _getPositionManager());
    _;
}
```

Recommendation:

Rename to `onlyPositionManager`, emit clear errors on misconfiguration, and expose the position manager address via a public getter.

Updates

The team resolved the issue by changing the name of the function.

SHB.21 Payable Functions Ignore Msg.Value

- Severity : INFORMATIONAL
- Status : Fixed
- Likelihood : 0
- Impact : 1

Description:

Certain deposit and withdraw functions accept Ether but neither use nor refund `msg.value`, leading to accidental ETH loss or unexpected balances.

Recommendation:

Reject unintended Ether by requiring `msg.value == 0` or handle native asset flows explicitly.

Updates

The team resolved the issue by adding a check on msg.value.

SHB.22 Underflow Risk In Full-Withdraw Preview

- Severity : INFORMATIONAL
- Status : Fixed
- Likelihood : 0
- Impact : 1

Description:

_previewWithdrawAll subtracts totalBorrow from previewSwapCollateralToBorrow(totalCollateral) without ensuring the latter is larger, causing underflow and reverts in preview mode.

Files Affected:

SHB.22.1: Withdrawal Preview

```
borrowTokensRemaining = previewSwapCollateralToBorrow(totalCollateral) - totalBorrow;
```

Recommendation:

Return zero or revert with a clear error when converted collateral is insufficient to cover debt.

Updates

The team resolved the issue by checking if collateralInBorrowToken is less than totalBorrow.

4 Best Practices

BP.1 Use SafeERC20 methods instead of raw ERC20 operations

Description:

Directly calling `approve` or `transfer` bypasses SafeERC20's safety checks and may revert on non-standard ERC20 tokens. Using SafeERC20's `forceApprove`/`safeTransfer` avoids repeated allowance resets and improves compatibility with tokens that require zeroing allowances first, saving ∼5k gas per call and preventing stuck approvals.

Files Affected:

BP.1.1: src/positions/PositionManager.sol

```
function _deposit(
address vault,
uint64 transactionCode,
address _tokenAddress,
uint256 _value,
bytes calldata extras
)
internal
{
SafeERC20.safeTransferFrom(IERC20(_tokenAddress), msg.sender, address(this), _value);
IERC20(_tokenAddress).approve(vault, _value);
IUserVault(vault).deposit(msg.sender, transactionCode, _tokenAddress, _value, extras);
}
```

BP.1.2: src/strategies/positions/base/BorrowLending/Common/BaseBorrowLendingStrategy.sol

```
function _withdrawBorrow(
address user,
address _tokenAddress,
uint256 _value,
IVaultManager.VaultStrategyConfig calldata userConfig
)
internal
{
if (_tokenAddress != _getBorrowToken()) revert InvalidToken(
    ↪ _tokenAddress);
_drawDebt(user, 0, _value);

ERC20 token = ERC20(_getBorrowToken());
token.transfer(userConfig.user, token.balanceOf(user));
}
function _withdrawCollateral(
address user,
address _tokenAddress,
uint256 _value,
IVaultManager.VaultStrategyConfig calldata userConfig
)
internal
{
if (_tokenAddress != _getCollateralToken()) revert InvalidToken(
    ↪ _tokenAddress);
_repayDebt(user, _value, 0, userConfig.user);

ERC20 token = ERC20(_getCollateralToken());
token.transfer(userConfig.user, token.balanceOf(user));
}
```

BP.1.3: src/strategies/positions/base/BorrowLending/Morpho/BaseMorphoUtils.sol

```
if (collateralToAdd > 0) {
ERC20 collateral_token = ERC20(_getCollateralToken());
45 collateral_token.approve(address(morpho), collateralToAdd);
46 morpho.supplyCollateral(params, collateralToAdd, user, hex"");
47 }
...
82 ERC20(_getBorrowToken()).approve(address(morpho), debtToRepay);
83 (debtRepaid,) = morpho.repay(params, debtToRepay, 0, user, hex"");
```

BP.1.4: src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableStrategy.sol

```
204 function onFlashLoan(
205 address user,
206 uint64 transactionCode,
207 address _tokenAddress,
208 uint256 _value,
209 uint256 flashLoanAmount
210 )
...
225 ERC20(_getBorrowToken()).approve(_getFlashLoanCaller(), flashLoanAmount);
226 }
```

Recommendation:

Replace raw calls with SafeERC20 helpers. Example:

```
- IERC20(_tokenAddress).approve(vault, _value);
+ IERC20(_tokenAddress).forceApprove(vault, _value);

- token.transfer(userConfig.user, token.balanceOf(user));
+ IERC20(address(token)).safeTransfer(userConfig.user, token.balanceOf(user));
```

Apply analogous changes where approvals or transfers are performed.

BP.2 Avoid runtime keccak for initializer selector

Description:

Computing a function selector with keccak256 at runtime wastes gas (~200 gas) and increases bytecode. Using the `.selector` constant or `abi.encodeCall` performs the computation at compile time.

Files Affected:

BP.2.1: src/positions/VaultRegistry.sol

```
// Encode initializer for UserVault
bytes memory initData = abi.encodeWithSelector(bytes4(keccak256("initialize(address)")), vaultManager);

// Deploy a BeaconProxy
BeaconProxy proxy = new BeaconProxy(vaultBeacon, initData);
```

Recommendation:

Use the interface selector or `abi.encodeCall`:

```
- bytes memory initData = abi.encodeWithSelector(bytes4(keccak256("initialize(address)")), vaultManager);
+ bytes memory initData = abi.encodeWithSelector(IUserVault.initialize.selector, vaultManager);
```

or

```
+ bytes memory initData = abi.encodeCall(IUserVault.initialize, (vaultManager));
```

BP.3 Cache struct pointers and remove helper in VaultManager

Description:

Repeated mapping lookups incur multiple SLOADs (∼2,100 gas). Creating a temporary storage reference and constructing structs in memory removes redundant reads and makes _emptyReserved unnecessary.

Files Affected:

BP.3.1: src/positions/VaultManager.sol

```
48 function _emptyReserved() internal pure returns (uint256[10] memory empty) {
49 // memory arrays are zero-initialized by default
50 }
...
72 function getVaultStrategyConfig(address vault) external view returns (VaultStrategyConfig memory) {
73 if (vaultConfigs[vault].strategy == address(0)) revert VaultNotRegistered();
74 return vaultConfigs[vault];
75 }
77 function getDepositEnabled(address vault) external view returns (bool) {
78 if (vaultConfigs[vault].strategy == address(0)) revert VaultNotRegistered();
79 return vaultConfigs[vault].depositEnabled && globalDepositEnabled;
80 }
...
100 vaultConfigs[vault] = VaultStrategyConfig({
101 strategy: strategy,
102 lendingThreshold: lendingThreshold,
103 iteration: iteration,
104 depositEnabled: true,
105 user: user,
106 __reserved: _emptyReserved()
107 });
118 function upsertDepositEnabled(address vault, bool flag) external override onlyOwner {
119 if (vaultConfigs[vault].strategy == address(0)) revert VaultNotRegistered();
120 vaultConfigs[vault].depositEnabled = flag;
121
122 emit VaultDepositFlagUpdated(vault, _msgSender(), flag);
123 }
```

Recommendation:

Use storage references and remove _emptyReserved:

```
- function _emptyReserved() internal pure returns (uint256[10] memory empty) {}
...
- vaultConfigs[vault] = VaultStrategyConfig({
- strategy: strategy,
- lendingThreshold: lendingThreshold,
- iteration: iteration,
- depositEnabled: true,
- user: user,
- __reserved: _emptyReserved()
- });
+ VaultStrategyConfig storage cfg = vaultConfigs[vault];
+ cfg.strategy = strategy;
+ cfg.lendingThreshold = lendingThreshold;
+ cfg.iteration = iteration;
+ cfg.depositEnabled = true;
+ cfg.user = user;
```

And in read/update functions cache:

```
- if (vaultConfigs[vault].strategy == address(0)) revert
    ↪ VaultNotRegistered();
- return vaultConfigs[vault].depositEnabled && globalDepositEnabled;
+ VaultStrategyConfig storage cfg = vaultConfigs[vault];
+ if (cfg.strategy == address(0)) revert VaultNotRegistered();
+ return cfg.depositEnabled && globalDepositEnabled;
```

BP.4 Return abi.encode directly

Description:

Creating a temporary `bytes` variable before returning `abi.encode` increases bytecode size (∼20 bytes per function) and slightly raises gas. Returning the encoding directly is cheaper.

Files Affected:

BP.4.1: src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableStrategy.sol

```
84 function encodeConfigForInitialDeposit(
85 uint256 minimumCollateralWithSlippage,
86 uint256 leverageInWAD
87 )
88 external
89 pure
90 returns (bytes memory)
91 {
92 bytes memory config = abi.encode(minimumCollateralWithSlippage,
    ↪ leverageInWAD);
93 return config;
94 }
```

Recommendation:

```
- bytes memory config = abi.encode(minimumCollateralWithSlippage, leverageInWAD);
- return config;
+ return abi.encode(minimumCollateralWithSlippage, leverageInWAD);
```

Apply similarly to other encoding helpers.

BP.5 Remove unused variable in flash-loan withdrawal

Description:

`totalBorrow` is loaded and modified without being used, wasting gas and bytes.

Files Affected:

BP.5.1: src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableUtils.sol

```
244 uint256 totalBorrow = borrowBalance(user);
247 totalBorrow -= flashLoanValue;
```

Recommendation:

```
- uint256 totalBorrow = borrowBalance(user);
- totalBorrow -= flashLoanValue;
+ borrowBalance(user); // Optional: ensure view call if needed
```

BP.6 Cache router and use unchecked subtraction in swap helpers

Description:

Repeated `_getRouter()` calls and checked arithmetic add extra gas.

Files Affected:

BP.6.1: src/strategies/positions/base/Spectra/BaseSpectraUtils.sol

```
30 IRouter router = _getRouter();
31 ERC20(_tokenAddress).approve(address(router), _value);
...
37 return _getPT().balanceOf(user) - initialBalance;
```

Recommendation:

```
+ uint256 finalBalance = _getPT().balanceOf(user);
+ unchecked { return finalBalance - initialBalance; }
```

BP.7 Remove unused import and cache token references in price feed

Description:

`RouterUtil` is imported but never used; repeated calls to `_getPTToken()` and `_getIBTToken()` cause redundant reads.

BP.8 Simplify address and decimals handling in sBOLD price feed

Description:

Remove redundant casting, cache decimals.

BP.9 Remove redundant return statements

Description:

Calling internal void functions with `return` is unnecessary.

```diff
- return _depositCollateral(user, _tokenAddress, _value);
+ _depositCollateral(user, _tokenAddress, _value);
```

5 Conclusion

In this audit, we examined the design and implementation of Leveraged Looping Strategies contracts and discovered several issues of varying severity. Raga Finance team addressed all the issues raised in the initial report and implemented the necessary fixes.

However Shellboxes' auditors advised Raga Finance Team to maintain a high level of vigilance and participate in bounty programs in order to avoid any future complications.

6 Scope Files

6.1 Audit

Files	MD5 Hash
src/strategies/positions/implementation/Spectra/SpectraBOLDStrategy/SpectraBOLDStrategy.sol	90a99062ac24a0f1dbf1bf9a6bbfae7a
src/strategies/positions/implementation/Spectra/SpectraBOLDStrategy/SpectraBOLDUtils.sol	e81e68fadb4c96e1c61fa58f42e09a4f
src/strategies/positions/implementation/PriceFeed/SpectraBOLD/BOLDPriceFeed.sol	500a071a9f6d0587ab22a3476bb4a2d5
src/strategies/positions/implementation/PriceFeed/SpectraBOLD/sBOLDPriceFeed.sol	6ee11fca5964ddf2e025c026931680e1
src/strategies/positions/implementation/PriceFeed/SpectraBOLD/SpectrasBOLDPriceFeed.sol	df67961ccfb3d11525193aee32ab56b4
src/strategies/positions/implementation/PriceFeed/Common/SpectraPTIBTPriceFeed.sol	af46cbf1af469c1e756d5c3cde03808a
src/strategies/positions/implementation/Looping/SpectraMorphoBOLDFlashLoanable/SpectraMorphoBOLDFlashLoanableStrategy.sol	a1e03084de35a321ae4d38e47ed19eca
src/strategies/positions/implementation/Looping/SpectraMorphoBOLDFlashLoanable/SpectraMorphoBOLDFlashLoanableUtils.sol	5ef3dc3fa262bcd8f104109514d1579a
src/strategies/positions/implementation/BorrowLending/MorphoPTsBOLDStrategy.sol	a01b45a19e88bc1edbb5374c5838efdb
src/strategies/positions/base/Spectra/BaseSpectraCommands.sol	753cba27c28b997112099b9710f0ec90

src/strategies/positions/base/Spectra/BaseSpectraStrategy.sol	4fa242243d291942c3d7af5d0b350eb5
src/strategies/positions/base/Spectra/BaseSpectraUtils.sol	61b6d7020891993d9e2725e606676dad
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableCommon.sol	5f2f8adf44ab53cb71d7b08f9d5cff6f
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanablePreviewUtils.sol	e4d03306c2573504e969bdb8dfecd13f
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableStrategy.sol	03ba679b80edf5067ac063074a1b2327
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableUtils.sol	7b0b1b9551801b3c0071f17243a31d27
src/strategies/positions/base/BorrowLending/Morpho/BaseMorphoCommons.sol	a78be2e6d1ed44f6dadad1391af8fed1
src/strategies/positions/base/BorrowLending/Morpho/BaseMorphoUtils.sol	76b4f6c1efb0cb140237113139ddedd4
src/strategies/positions/base/BorrowLending/Common/BaseBorrowLendingStrategy.sol	e92e2197059bddd46327324e024c7a55
src/positions/Constants.sol	72e5cd9406a0378f60e25a6785cb574b
src/positions/PositionManager.sol	c4cc1664281b1d57afc09b3b9103ed34
src/positions/StrategyManager.sol	b4b246875b793272500f1501156a046f
src/positions/UserVault.sol	7ff5b7f511a10415109cd4de0d728e6b
src/positions/VaultBeacon.sol	8a0c691d58fce1d7b1ed960b3e0dfe7d

src/positions/VaultManager.sol	f0e80e57246a6f232a9af323c356657d
src/positions/VaultRegistry.sol	88a06a533b27a9ad02ba8cff374ff2ef
src/libraries/LoopingMathLib.sol	b7cafb5dea062c08c94039e538bfc962
src/libraries/LoopingUtilStorage.sol	9166bfee741589c0a0769f8d9bb766d3

6.2 Re-Audit

Files	MD5 Hash
src/strategies/positions/implementation/Spectra/SpectraBOLDStrategy/SpectraBOLDStrategy.sol	90a99062ac24a0f1dbf1bf9a6bbfae7a
src/strategies/positions/implementation/Spectra/SpectraBOLDStrategy/SpectraBOLDUtils.sol	e81e68fadb4c96e1c61fa58f42e09a4f
src/strategies/positions/implementation/PriceFeed/SpectraBOLD/BOLDPriceFeed.sol	a99b12fa63661901b10dd0c73a077792
src/strategies/positions/implementation/PriceFeed/SpectraBOLD/sBOLDPriceFeed.sol	6ee11fca5964ddf2e025c026931680e1
src/strategies/positions/implementation/PriceFeed/SpectraBOLD/SpectrasBOLDPriceFeed.sol	df67961ccfb3d11525193aee32ab56b4
src/strategies/positions/implementation/PriceFeed/Common/SpectraPTIBTPriceFeed.sol	af46cbf1af469c1e756d5c3cde03808a
src/strategies/positions/implementation/Looping/SpectraMorphoBOLDFlashLoanable/SpectraMorphoBOLDFlashLoanableStrategy.sol	a1e03084de35a321ae4d38e47ed19eca

src/strategies/positions/implementation/Looping/SpectraMorphoBOLDFlashLoanable/SpectraMorphoBOLDFlashLoanableUtils.sol	7cdb52196ed3f1ce768e5e0e43137e26
src/strategies/positions/implementation/BorrowLending/MorphoPTsBOLDStrategy.sol	a01b45a19e88bc1edbb5374c5838efdb
src/strategies/positions/base/Spectra/BaseSpectraCommands.sol	753cba27c28b997112099b9710f0ec90
src/strategies/positions/base/Spectra/BaseSpectraStrategy.sol	f3d9dc05b16cd8c6d33c317ee0e48143
src/strategies/positions/base/Spectra/BaseSpectraUtils.sol	5ed18bf9055f38a5a0accde0b7a61be9
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableCommon.sol	addd10e462c403481687fdae719505cf
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanablePreviewUtils.sol	561140fc12b4d2815fab4f1ecc0ff149
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableStrategy.sol	e5e9dca11cb66c0f5b35e2ce800819f0
src/strategies/positions/base/Looping/SpectraFlashLoanable/BaseSpectraFlashLoanableUtils.sol	85c51b7cd08200ae5dcc94c3bdc04f2b
src/strategies/positions/base/BorrowLending/Morpho/BaseMorphoCommons.sol	a78be2e6d1ed44f6dadad1391af8fed1
src/strategies/positions/base/BorrowLending/Morpho/BaseMorphoUtils.sol	1d70263f30f82c258774cc2e4bb11c99
src/strategies/positions/base/BorrowLending/Common/BaseBorrowLendingStrategy.sol	e713e6f29953b03ef46432032e4b0a88

src/positions/Constants.sol	4db50db6cb58d89e0495ae705a4cf96f
src/positions/PositionManager.sol	10a6bf60f0a7996bbee1c04b385ea2b1
src/positions/StrategyManager.sol	9f96fe63665f6b154b728ed587f7f264
src/positions/UserVault.sol	a88330ad50edfcae7db8ab9b65a1eba5
src/positions/VaultBeacon.sol	8a0c691d58fce1d7b1ed960b3e0dfe7d
src/positions/VaultManager.sol	e11dccc9fb8bbfc41fc46d99a3f78cd8
src/positions/VaultRegistry.sol	1d02bc76a9e216884aad74041210b20c
src/libraries/LoopingMathLib.sol	14552e5535f9fdff4109931460f72989
src/libraries/LoopingUtilStorage.sol	9166bfee741589c0a0769f8d9bb766d3

7 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology’s proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don’t offer any kind of investing advice and shouldn’t be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com