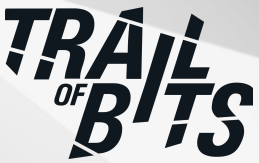

Spool V2

Security Assessment

May 9, 2023

Prepared for:
Spool DAO

Prepared by: **Simone Monica, Guillermo Larregay, Vara Prasad Bandaru, and Tarun Bansal**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

The Spool DAO engaged Trail of Bits to review the security of its Spool V2 codebase. From February 13 to March 20, 2022, a team of four consultants conducted a security review of the client-provided source code, with eight person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed automated and manual testing of the target system and its codebase.

Summary of Findings

The audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	5
Medium	14
Low	12
Informational	5
Undetermined	2

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	2
Auditing and Logging	1
Data Validation	17
Testing	1
Undefined Behavior	17

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-SPL-2**
  The `grantSmartVaultOwnership` function can be used to cause a DoS of the smart vault creation process.

- **TOB-SPL-9**
  All user deposits are not deposited to strategies due to incorrect bookkeeping.

- **TOB-SPL-14**
  The `doHardWork` function uses the incorrect variable for exchange rates.

- **TOB-SPL-26**
  The reallocation process reverts on smart vaults with ghost strategies.

- **TOB-SPL-31**
  Users who deposit and then withdraw assets before `doHardWork` is called will receive zero tokens from their withdrawal operations.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineers were associated with this project:

Simone Monica, Consultant
simone.monica@trailofbits.com

Guillermo Larregay, Consultant
guillermo.larregay@trailofbits.com

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Vara Prasad Bandaru, Consultant
vara.bandaru@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
February 9, 2023	Pre-project kickoff call
February 17, 2023	Status update meeting #1
February 24, 2023	Status update meeting #2
March 10, 2023	Status update meeting #3
March 20, 2023	Delivery of report draft
March 20, 2023	Report readout meeting
May 9, 2023	Delivery of final report with fix review appendix

Project Goals

The engagement was scoped to provide a security assessment of the Spool DAO's Spool V2 codebase. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can an attacker steal or freeze funds?
- Are there appropriate access control measures in place for the different roles in the system?
- Does the system's behavior match the specification?
- Can users lose funds due to the system's mechanisms, such as the smart vault flushing, "do hard work," depositing, and withdrawing mechanisms?
- Are users' deposited funds correctly deployed in the different strategies?
- Can a smart vault be bricked?
- Do the allocator providers return the correct fund allocations?
- Can the system's operations unexpectedly revert?
- Are the platform fees correctly computed?
- Are admins able to upgrade the contracts when required?
- Do operations for minting, burning, and transferring tokens and NFTs work as expected?
- Does removing a strategy break the system?

Project Targets

The engagement involved a review and testing of the following target.

Spool V2

Repository	https://github.com/SpoolFi/spool-v2-core
Version	`8f90a7d7b930da80cabcfab6d5049d4d69e67c00`
Type	Solidity
Platform	Ethereum

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- **General:** We reviewed the entire codebase for common Solidity flaws, such as missing contract existence checks on low-level calls, issues with access controls, issues related to the protocol's pausing functionality, and issues related to upgradeable contracts, like state variables that are set in the implementation's constructor.

- **Providers:** The `providers` folder contains three allocators that compute the allocations between the different strategies. We manually reviewed the folder for possible errors in the calculator, such as division-by-zero errors. Additionally, we looked for inconsistencies due to the presence of ghost strategies in smart vaults.

- **Rewards:** The `rewards` folder allows smart vault owners to provide additional token incentives. We manually reviewed the code that configures rewards for appropriate validation to prevent errors that could result in the loss of funds; we also checked for the correct use of Merkle proofs to claim rewards.

- **Smart vault creation:** The system allows anyone to make a smart vault through the `SmartVaultFactory` contract. We manually checked that the smart vault configuration is correctly validated according to the documentation and checked for ways to cause a DoS of the smart vault creation process.

- **Smart vault guards and actions:** Smart vaults can have guards and actions that run when users perform specific actions, such as depositing or withdrawing. They are set during the smart vault creation process. We manually reviewed the process for ways that guards and actions might fail. We checked that the guard manager correctly calls the guard contracts with various input and output types.

- **Core smart vault contracts:** The core smart vault contracts allow users to deposit into and withdraw from a smart vault. Funds deposited into a smart vault are managed by the smart vault and are deposited into and withdrawn from the strategies set. Additionally, a smart vault's funds could be reallocated by a privileged user. We checked for issues that could occur when users deposit and withdraw, such as those that could allow users to avoid paying platform fees and lose deposited funds, and we checked that shares are correctly withdrawn. We looked for ways in which flushing and syncing operations may incorrectly account for deposits or withdrawals. We analyzed the "do hard work" functionality for bookkeeping issues that could result in fewer deposits in strategies than expected, and for opportunities to conduct external attacks, such as front-running and

back-running when depositing into or withdrawing from a strategy. Finally, we reviewed the correctness of the reallocation process. We looked for edge cases pertaining to smart vaults that contain one or more ghost strategies. We also looked for rounding errors and the possibility of dust tokens at the end of reallocation.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We performed only a light review of the strategies added by PR #174. We identified one related finding during this review (TOB-SPL-25), but the strategies will need another deeper review.
- Third-party components such as Compound were out of scope, so they were not reviewed.
- We assumed that the implemented mathematical formulas in the allocation providers are correct, and we did not review them.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Slither	A static analysis framework that can statically verify algebraic relationships between Solidity variables	Appendix D: Slither
Necessist	A framework that runs tests with statements and method calls removed to help identify broken tests	Appendix D: Necessist

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The codebase relies heavily on arithmetic calculations to calculate strategies, allocations, fees, and rewards. However, we found some issues involving incorrect arithmetic (TOB-SPL-16), the use of unchecked blocks that are not documented, equations missing from the documentation, and broken and insufficient testing (TOB-SPL-27). Additionally, we found some unsafe casts that may cause issues (TOB-SPL-38).	**Weak**
Auditing	Certain state-changing functions do not emit events (TOB-SPL-32). The documentation does not indicate how events are monitored off-chain to detect abnormal behavior. The Spool team is working on an incident response plan; however, it was not ready at the time of the audit.	**Moderate**
Authentication / Access Controls	The system implements different roles with different privileges that do not overlap, and the documentation specifies each role's privileges. However, although there are some access control checks, some are also missing; additionally, some roles are not assigned correctly. The test suite provided does not have specific tests for roles and privileges. Given the complex roles of the system, the documentation should also indicate which roles each contract contains, as well as when and by who they are granted.	**Moderate**
Complexity Management	In general, the code is complex and hard to follow. There are multiple instances of nested loops, up to three levels deep in some cases. Often data structures are not used	**Weak**

	intuitively or consistently; for example, some functions that deal with arrays that should be the same length increase the length of one array to store unrelated data. Modifiers and `require` statements are used inconsistently in the codebase to check for permissions.	
Decentralization	The team considers Spool V2 a "decentralized middleware." However, there are certain roles that are allowed to change system parameters at any time, and some contracts require a form of whitelisting before they can be used. Additionally, all contracts are upgradeable by the team, and users cannot opt out of upgrades.	**Weak**
Documentation	The documentation is a high-level specification of how the system works and includes diagrams for use cases. Some parts of the documentation are not complete or up to date with the implementation. The source code uses NatSpec documentation comments, but not all functions are commented, and some functions' inline comments lack detail.	**Weak**
Transaction Ordering Risks	Some of the system's actions are exposed to MEV issues, such as depositing into, withdrawing from, and reallocating strategies. The sender of such transactions must accurately set the slippage, otherwise a significant number of funds could be lost. Consider calling the `doHardWork` function multiple times a day if the amount in a given deposit or withdrawal operation is above a certain limit to prevent significant slippage and MEV exposure.	**Moderate**
Low-Level Manipulation	One of the contracts manually constructs the calldata for external calls and decodes the return data. The function implements manual encoding and decoding of common data types, which could cause issues (TOB-SPL-11). Only two functions use low-level calls, but none of them check for contract existence (TOB-SPL-13). Also, the parts of the codebase that use low-level manipulation lack sufficient documentation.	**Moderate**

Testing and Verification	The provided test suite does not test all functions and normal use cases. Some tests are incorrectly designed or have typographical errors. Some of the mock contracts are not fully implemented and lack certain checks.	**Weak**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Solidity compiler optimizations can be problematic	Undefined Behavior	**Undetermined**
2	Risk of SmartVaultFactory DoS due to lack of access controls on grantSmartVaultOwnership	Access Controls	**High**
3	Lack of zero-value check on constructors and initializers	Data Validation	**Medium**
4	Upgradeable contracts set state variables in the constructor	Data Validation	**Medium**
5	Insufficient validation of oracle price data	Data Validation	**Low**
6	Incorrect handling of fromVaultsOnly in removeStrategy	Data Validation	**Low**
7	Risk of LinearAllocationProvider and ExponentialAllocationProvider reverts due to division by zero	Data Validation	**Medium**
8	Strategy APYs are never updated	Undefined Behavior	**Medium**
9	Incorrect bookkeeping of assets deposited into smart vaults	Undefined Behavior	**High**
10	Risk of malformed calldata of calls to guard contracts	Data Validation	**Low**
11	GuardManager does not account for all possible types when encoding guard arguments	Undefined Behavior	**Low**

12	Use of encoded values in guard contract comparisons could lead to opposite results	Undefined Behavior	**Low**
13	Lack of contract existence checks on low-level calls	Data Validation	**Low**
14	Incorrect use of exchangeRates in doHardWork	Undefined Behavior	**High**
15	LinearAllocationProvider could return an incorrect result	Undefined Behavior	**Medium**
16	Incorrect formula used for adding/subtracting two yields	Undefined Behavior	**Medium**
17	Smart vaults with re-registered strategies will not be usable	Undefined Behavior	**Low**
18	Incorrect handling of partially burned NFTs results in incorrect SVT balance calculation	Undefined Behavior	**Low**
19	Transfers of D-NFTs result in double counting of SVT balance	Data Validation	**Medium**
20	Flawed loop for syncing flushes results in higher management fees	Data Validation	**Medium**
21	Incorrect ghost strategy check	Data Validation	**Informational**
22	Reward configuration not initialized properly when reward is zero	Undefined Behavior	**Low**
23	Missing function for removing reward tokens from the blacklist	Undefined Behavior	**Informational**
24	Risk of unclaimed shares due to loss of precision in reallocation operations	Undefined Behavior	**Informational**
25	Curve3CoinPoolAdapter's _addLiquidity reverts due to incorrect amounts deposited	Undefined Behavior	**Medium**

26	Reallocation process reverts when a ghost strategy is present	Data Validation	**High**
27	Broken test cases that hide security issues	Testing	**Informational**
28	Reward emission can be extended for a removed reward token	Data Validation	**Medium**
29	A reward token cannot be added once it is removed from a smart vault	Data Validation	**Low**
30	Missing whenNotPaused modifier	Undefined Behavior	**Low**
31	Users who deposit and then withdraw before doHardWork lose their tokens	Undefined Behavior	**High**
32	Lack of events emitted for state-changing functions	Auditing and Logging	**Informational**
33	Removal of a strategy could result in loss of funds	Undefined Behavior	**Medium**
34	ExponentialAllocationProvider reverts on strategies without risk scores	Data Validation	**Medium**
35	Removing a strategy makes the smart vault unusable	Data Validation	**Medium**
36	Issues with the management of access control roles in deployment script	Access Controls	**Low**
37	Risk of DoS due to unbounded loops	Data Validation	**Medium**
38	Unsafe casts throughout the codebase	Data Validation	**Undetermined**

Detailed Findings

1. Solidity compiler optimizations can be problematic	
Severity: **Undetermined**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-SPL-1
Target: `foundry.toml`	

Description

Spool V2 has enabled optional compiler optimizations in Solidity.

There have been several optimization bugs with security implications. Moreover, optimizations are actively being developed. Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised.

High-severity security issues due to optimization bugs have occurred in the past. A high-severity bug in the `emscripten`-generated `solc-js` compiler used by Truffle and Remix persisted until late 2018. The fix for this bug was not reported in the Solidity CHANGELOG. Another high-severity optimization bug resulting in incorrect bit shift results was patched in Solidity 0.5.6. More recently, another bug due to the incorrect caching of `keccak256` was reported.

A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe.

It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations.

Exploit Scenario

A latent or future bug in Solidity compiler optimizations—or in the Emscripten transpilation to `solc-js`—causes a security vulnerability in the Spool V2 contracts.

Recommendations

Short term, measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug.

Long term, monitor the development and adoption of Solidity compiler optimizations to assess their maturity.

2. Risk of SmartVaultFactory DoS due to lack of access controls on grantSmartVaultOwnership

Severity: **High**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-SPL-2
Target: `SmartVaultFactory.sol`, `access/SpoolAccessControl.sol`	

Description

Anyone can set the owner of the next smart vault to be created, which will result in a DoS of the `SmartVaultFactory` contract.

The `grantSmartVaultOwnership` function in the `SpoolAccessControl` contract allows anyone to set the owner of a smart vault. This function reverts if an owner is already set for the provided smart vault.

```
function grantSmartVaultOwnership(address smartVault, address owner) external {
    if (smartVaultOwner[smartVault] != address(0)) {
        revert SmartVaultOwnerAlreadySet(smartVault);
    }

    smartVaultOwner[smartVault] = owner;
}
```

Figure 2.1: The `grantSmartVaultOwnership` function in `SpoolAccessControl.sol`

The `SmartVaultFactory` contract implements two functions for deploying new smart vaults: the `deploySmartVault` function uses the `create` opcode, and the `deploySmartVaultDeterministically` function uses the `create2` opcode. Both functions create a new smart vault and call the `grantSmartVaultOwnership` function to make the message sender the owner of the newly created smart vault.

Any user can pre-compute the address of the new smart vault for a `deploySmartVault` transaction by using the address and nonce of the `SmartVaultFactory` contract; to compute the address of the new smart vault for a `deploySmartVaultDeterministically` transaction, the user could front-run the transaction to capture the salt provided by the user who submitted it.

Exploit Scenario

Eve pre-computes the address of the new smart vault that will be created by the `deploySmartVault` function in the `SmartVaultFactory` contract. She then calls the `grantSmartVaultOwnership` function with the pre-computed address and a nonzero

address as arguments. Now, every call to the `deploySmartContract` function reverts, making the `SmartVaultFactory` contract unusable.

Using a similar strategy, Eve blocks the `deploySmartVaultDeterministically` function by front-running the user transaction to set the owner of the smart vault address computed using the user-provided salt.

Recommendations

Short term, add the `onlyRole(ROLE_SMART_VAULT_INTEGRATOR, msg.sender)` modifier to the `grantSmartVaultOwnership` function to restrict access to it.

Long term, follow the principle of least privilege by restricting access to the functions that grant specific privileges to actors of the system.

3. Lack of zero-value check on constructors and initializers

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-3
Target: Several contracts	

Description

Several contracts' constructors and initialization functions fail to validate incoming arguments. As a result, important state variables could be set to the zero address, which would result in the loss of assets.

```
constructor(
    ISpoolAccessControl accessControl_,
    IAssetGroupRegistry assetGroupRegistry_,
    IRiskManager riskManager_,
    IDepositManager depositManager_,
    IWithdrawalManager withdrawalManager_,
    IStrategyRegistry strategyRegistry_,
    IMasterWallet masterWallet_,
    IUsdPriceFeedManager priceFeedManager_,
    address ghostStrategy
) SpoolAccessControllable(accessControl_) {
    _assetGroupRegistry = assetGroupRegistry_;
    _riskManager = riskManager_;
    _depositManager = depositManager_;
    _withdrawalManager = withdrawalManager_;
    _strategyRegistry = strategyRegistry_;
    _masterWallet = masterWallet_;
    _priceFeedManager = priceFeedManager_;
    _ghostStrategy = ghostStrategy;
}
```

Figure 3.1: The `SmartVaultManager` contract's `constructor` function in `spool-v2-core/SmartVaultManager.sol#L111–L130`

These constructors include that of the `SmartVaultManager` contract, which sets the `_masterWallet` address (figure 3.1). `SmartVaultManager` contract is the entry point of the system and is used by users to deposit their tokens. User deposits are transferred to the `_masterWallet` address (figure 3.2).

```
function _depositAssets(DepositBag calldata bag) internal returns (uint256) {
    [...]
    for (uint256 i; i < deposits.length; ++i) {
        IERC20(tokens[i]).safeTransferFrom(msg.sender, address(_masterWallet),
```

```
deposits[i]);
    }
    [...]
}
```

Figure 3.2: The `_depositAssets` function in spool-v2-core/SmartVaultManager.sol#L649–L676

If `_masterWallet` is set to the zero address, the tokens will be transferred to the zero address and will be lost permanently.

The constructors and initialization functions of the following contracts also fail to validate incoming arguments:

- `StrategyRegistry`
- `DepositSwap`
- `SmartVault`
- `SmartVaultFactory`
- `SpoolAccessControllable`
- `DepositManager`
- `RiskManager`
- `SmartVaultManager`
- `WithdrawalManager`
- `RewardManager`
- `RewardPool`
- `Strategy`

Exploit Scenario

Bob deploys the Spool system. During deployment, Bob accidentally sets the `_masterWallet` parameter of the `SmartVaultManager` contract to the zero address. Alice, excited about the new protocol, deposits 1 million WETH into it. Her deposited WETH tokens are transferred to the zero address, and Alice loses 1 million WETH.

Recommendations

Short term, add zero-value checks on all constructor arguments to ensure that the deployer cannot accidentally set incorrect values.

Long term, use Slither, which will catch functions that do not have zero-value checks.

4. Upgradeable contracts set state variables in the constructor

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-SPL-4
Target: `managers/RewardManager.sol`, `managers/RewardPool.sol`, `strategies/Strategy.sol`	

Description

The state variables set in the constructor of the `RewardManager` implementation contract are not visible in the proxy contract, making the `RewardManager` contract unusable. The same issue exists in the `RewardPool` and `Strategy` smart contracts.

Upgradeable smart contracts using the `delegatecall` proxy pattern should implement an initializer function to set state variables in the proxy contract storage. The `constructor` function can be used to set immutable variables in the implementation contract because these variables do not consume storage slots and their values are inlined in the deployed code.

The `RewardManager` contract is deployed as an upgradeable smart contract, but it sets the state variable `_assetGroupRegistry` in the `constructor` function.

```
contract RewardManager is IRewardManager, RewardPool, ReentrancyGuard {
    ...

    /* ========== STATE VARIABLES ========== */

    /// @notice Asset group registry
    IAssetGroupRegistry private _assetGroupRegistry;

    ...

    constructor(
        ISpoolAccessControl spoolAccessControl,
        IAssetGroupRegistry assetGroupRegistry_,
        bool allowPoolRootUpdates
    ) RewardPool(spoolAccessControl, allowPoolRootUpdates) {
        _assetGroupRegistry = assetGroupRegistry_;
    }
```

Figure 4.1: The `constructor` function in `spool-v2-core/RewardManager.sol`

The value of the `_assetGroupRegistry` variable will not be visible in the proxy contract, and the admin will not be able to add reward tokens to smart vaults, making the `RewardManager` contract unusable.

The following smart contracts are also affected by the same issue:

1. The `ReentrancyGuard` contract, which is non-upgradeable and is extended by `RewardManager`
2. The `RewardPool` contract, which sets the state variable `allowUpdates` in the constructor
3. The `Strategy` contract, which sets the state variable `StrategyName` in the constructor

Exploit Scenario

Bob creates a smart vault and wants to add a reward token to it. He calls the `addToken` function on the `RewardManager` contract, but the transaction unexpectedly reverts.

Recommendations

Short term, make the following changes:

1. Make `_assetGroupRegistry` an immutable variable in the `RewardManager` contract.
2. Extend the `ReentrancyGuardUpgradeable` contract in the `RewardManager` contract.
3. Make `allowUpdates` an immutable variable in the `RewardPool` contract.
4. Move the statement `_strategyName = strategyName_;` from the `Strategy` contract's constructor to the contract's `__Strategy_init` function.
5. Review all of the upgradeable contracts to ensure that they extend only upgradeable library contracts and that the inherited contracts have a `__gap` storage variable to prevent storage collision issues with future upgrades.

Long term, review all of the upgradeable contracts to ensure that they use the initializer function instead of the `constructor` function to set state variables. Use `slither-check-upgradeability` to find issues related to upgradeable smart contracts.

5. Insufficient validation of oracle price data

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-SPL-5
Target: `managers/UsdPriceFeedManager.sol`	

Description

The current validation of the values returned by Chainlink's `latestRoundData` function could result in the use of stale data.

The `latestRoundData` function returns the following values: the `answer`, the `roundId` (which represents the current round), the `answeredInRound` value (which corresponds to the round in which the answer was computed), and the `updatedAt` value (which is the timestamp of when the round was updated). An `updatedAt` value of `0` means that the round is not complete and should not be used. An `answeredInRound` value that is less than the `roundId` indicates stale data.

However, the `_getAssetPriceInUsd` function in the `UsdPriceFeeManager` contract does not check for these conditions.

```
function _getAssetPriceInUsd(address asset) private view returns (uint256
assetPrice) {
    (
        /* uint80 roundId */
        ,
        int256 answer,
        /* uint256 startedAt */
        ,
        /* uint256 updatedAt */
        ,
        /* uint80 answeredInRound */
    ) = assetPriceAggregator[asset].latestRoundData();

    if (answer < 1) {
        revert NonPositivePrice({price: answer});
    }

    return uint256(answer);
}
```

Figure 5.1: The `_getAssetPriceInUsd` function in spool-v2-core/UsdPriceFeedManager.sol#L99–L116

Exploit Scenario
The price of an asset changes, but the Chainlink price feed is not updated correctly. The system uses the stale price data, and as a result, the asset is not correctly distributed in the strategies.

Recommendations
Short term, have `_getAssetPriceInUsd` perform the following sanity check: `require(updatedAt != 0 && answeredInRound == roundId)`. This check will ensure that the round has finished and that the pricing data is from the current round.

Long term, when integrating with third-party protocols, make sure to accurately read their documentation and implement the appropriate sanity checks.

6. Incorrect handling of fromVaultsOnly in removeStrategy

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-SPL-6
Target: `managers/SmartVaultManager.sol`	

Description

The `removeStrategy` function allows Spool admins to remove a strategy from the smart vaults using it. Admins are also able to remove the strategy from the `StrategyRegistry` contract, but only if the value of `fromVaultsOnly` is `false`; however, the implementation enforces the opposite, as shown in figure 6.1.

```
function removeStrategy(address strategy, bool fromVaultsOnly) external {
    _checkRole(ROLE_SPOOL_ADMIN, msg.sender);
    _checkRole(ROLE_STRATEGY, strategy);

    ...

    if (fromVaultsOnly) {
        _strategyRegistry.removeStrategy(strategy);
    }
}
```

Figure 6.1: The `removeStrategy` function in spool-v2-core/SmartVaultManager.sol#L298–L317

Exploit Scenario

Bob, a Spool admin, calls `removeStrategy` with `fromVaultsOnly` set to `true`, believing that this call will not remove the strategy from the `StrategyRegistry` contract. However, once the transaction is executed, he discovers that the strategy was indeed removed.

Recommendations

Short term, replace `if (fromVaultsOnly)` with `if (!fromVaultsOnly)` in the `removeStrategy` function to implement the expected behavior.

Long term, improve the system's unit and integration tests to catch issues such as this one.

7. Risk of LinearAllocationProvider and ExponentialAllocationProvider reverts due to division by zero

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-7
Target: `providers/LinearAllocationProvider.sol`, `providers/ExponentialAllocationProvider.sol`	

Description

The `LinearAllocationProvider` and `ExponentialAllocationProvider` contracts' `calculateAllocation` function can revert due to a division-by-zero error: `LinearAllocationProvider`'s function reverts when the sum of the strategies' APY values is `0`, and `ExponentialAllocationProvider`'s function reverts when a single strategy has an APY value of `0`.

Figure 7.1 shows a snippet of the `LinearAllocationProvider` contract's `calculateAllocation` function; if the `apySum` variable, which is the sum of all the strategies' APY values, is `0`, a division-by-zero error will occur.

```
        uint8[] memory arrayRiskScores = data.riskScores;
         for (uint8 i; i < data.apys.length; ++i) {
             apySum += (data.apys[i] > 0 ? uint256(data.apys[i]) : 0);
             riskSum += arrayRiskScores[i];
         }

         uint8 riskt = uint8(data.riskTolerance + 10); // from 0 to 20

         for (uint8 i; i < data.apys.length; ++i) {
             uint256 apy = data.apys[i] > 0 ? uint256(data.apys[i]) : 0;
             apy = (apy * FULL_PERCENT) / apySum;
```

Figure 7.1: Part of the `calculateAllocation` function in spool-v2-core/LinearAllocationProvider.sol#L39–L49

Figure 7.2 shows that for the `ExponentialAllocationProvider` contract's `calculateAllocation` function, if the call to `log_2` occurs with `partApy` set to `0`, the function will revert because of `log_2`'s `require` statement shown in figure 7.3.

```
        for (uint8 i; i < data.apys.length; ++i) {
             uint256 uintApy = (data.apys[i] > 0 ? uint256(data.apys[i]) : 0);
             int256 partRiskTolerance = fromUint(uint256(riskArray[uint8(20 -
```

```
riskt)]));

            partRiskTolerance = div(partRiskTolerance, _100);
            int256 partApy = fromUint(uintApy);
            partApy = div(partApy, _100);

            int256 apy = exp_2(mul(partRiskTolerance, log_2(partApy)));
```

Figure 7.2: Part of the `calculateAllocation` function in spool-v2-core/ExponentialAllocationProvider.sol#L323–L331

```
    function log_2(int256 x) internal pure returns (int256) {
        unchecked {
            require(x > 0);
```

Figure 7.3: Part of the `log_2` function in spool-v2-core/ExponentialAllocationProvider.sol#L32–L34

Exploit Scenario

Bob deploys a smart vault with two strategies using the `ExponentialAllocationProvider` contract. At some point, one of the strategies has `0` APY, causing the transaction call to reallocate the assets to unexpectedly revert.

Recommendations

Short term, modify both versions of the `calculateAllocation` function so that they correctly handle cases in which a strategy's APY is `0`.

Long term, improve the system's unit and integration tests to ensure that the basic operations work as expected.

8. Strategy APYs are never updated

Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-8
Target: `managers/StrategyRegistry.sol`	

Description

The `_updateDhwYieldAndApy` function is never called. As a result, each strategy's APY will constantly be set to `0`.

```
   function _updateDhwYieldAndApy(address strategy, uint256 dhwIndex, int256
yieldPercentage) internal {
        if (dhwIndex > 1) {
            unchecked {
                int256 timeDelta = int256(block.timestamp -
_stateAtDhw[address(strategy)][dhwIndex - 1].timestamp);
                if (timeDelta > 0) {
                    int256 normalizedApy = yieldPercentage * SECONDS_IN_YEAR_INT /
timeDelta;
                    int256 weight = _getRunningAverageApyWeight(timeDelta);
                    _apys[strategy] =
                        (_apys[strategy] * (FULL_PERCENT_INT - weight) +
normalizedApy * weight) / FULL_PERCENT_INT;
                }
            }
        }
    }
```

Figure 8.1: The `_updateDhwYieldAndApy` function in spool-v2-core/StrategyManager.sol#L298–L317

A strategy's APY is one of the parameters used by an allocator provider to decide where to allocate the assets of a smart vault. If a strategy's APY is `0`, the `LinearAllocationProvider` and `ExponentialAllocationProvider` contracts will both revert when `calculateAllocation` is called due to a division-by-zero error.

```
       // set allocation
        if (uint16a16.unwrap(allocations) == 0) {
            _riskManager.setRiskProvider(smartVaultAddress,
specification.riskProvider);
            _riskManager.setRiskTolerance(smartVaultAddress,
specification.riskTolerance);
            _riskManager.setAllocationProvider(smartVaultAddress,
specification.allocationProvider);
```

```
            allocations = _riskManager.calculateAllocation(smartVaultAddress,
specification.strategies);
        }
```

Figure 8.2: Part of the `_integrateSmartVault` function, which is called when a vault is created, in `spool-v2-core/SmartVaultFactory.sol#L313–L320`

When a vault is created, the code in figure 8.2 is executed. For vaults whose `strategyAllocation` variable is set to `0`, which means the value will be calculated by the smart contract, and whose `allocationProvider` variable is set to the `LinearAllocationProvider` or `ExponentialAllocationProvider` contract, the creation transaction will revert due to a division-by-zero error. Transactions for creating vaults with a nonzero `strategyAllocation` and with the same `allocationProvider` values mentioned above will succeed; however, the fund reallocation operation will revert because the `_updateDhwYieldAndApy` function is never called, causing the strategies' APYs to be set to `0`, in turn causing the same division-by-zero error. Refer to finding TOB-SPL-7, which is related to this issue; even if that finding is fixed, incorrect results would still occur because of the missing `_updateDhwYieldAndApy` calls.

Exploit Scenario

Bob tries to deploy a smart vault with `strategyAllocation` set to `0` and `allocationProvider` set to `LinearAllocationProvider`. The transaction unexpectedly fails.

Recommendations

Short term, add calls to `_updateDhwYieldAndApy` where appropriate.

Long term, improve the system's unit and integration tests to ensure that the basic operations work as expected.

9. Incorrect bookkeeping of assets deposited into smart vaults

Severity: **High**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-9
Target: `managers/DepositManager.sol`	

Description

Assets deposited by users into smart vaults are incorrectly tracked. As a result, assets deposited into a smart vault's strategies when the `flushSmartVault` function is invoked correspond to the last deposit instead of the sum of all deposits into the strategies.

When depositing assets into a smart vault, users can decide whether to invoke the `flushSmartVault` function. A smart vault flush is a synchronization process that makes deposited funds available to be deployed into the strategies and makes withdrawn funds available to be withdrawn from the strategies. However, the internal bookkeeping of deposits keeps track of only the last deposit of the current flush cycle instead of the sum of all deposits (figure 9.1).

```
function depositAssets(DepositBag calldata bag, DepositExtras calldata bag2)
     external
     onlyRole(ROLE_SMART_VAULT_MANAGER, msg.sender)
     returns (uint256[] memory, uint256)
 {

    ...
    // transfer tokens from user to master wallet
     for (uint256 i; i < bag2.tokens.length; ++i) {
         _vaultDeposits[bag.smartVault][bag2.flushIndex][i] = bag.assets[i];
     }
    ...
```

Figure 9.1: A snippet of the `depositAssets` function in `spool-v2-core/DepositManager.sol#L379–L439`

The `_vaultDeposits` variable is then used to calculate the asset distribution in the `flushSmartVault` function.

```
function flushSmartVault(
     address smartVault,
     uint256 flushIndex,
     address[] calldata strategies,
     uint16a16 allocation,
     address[] calldata tokens
```

```
    ) external returns (uint16a16) {
        _checkRole(ROLE_SMART_VAULT_MANAGER, msg.sender);

        if (_vaultDeposits[smartVault][flushIndex][0] == 0) {
            return uint16a16.wrap(0);
        }

        // handle deposits
        uint256[] memory exchangeRates = SpoolUtils.getExchangeRates(tokens,
_priceFeedManager);
        _flushExchangeRates[smartVault][flushIndex].setValues(exchangeRates);

        uint256[][] memory distribution = distributeDeposit(
            DepositQueryBag1({
                deposit:
_vaultDeposits[smartVault][flushIndex].toArray(tokens.length),
                exchangeRates: exchangeRates,
                allocation: allocation,
                strategyRatios: SpoolUtils.getStrategyRatiosAtLastDhw(strategies,
_strategyRegistry)
            })
        );

        ...

        return _strategyRegistry.addDeposits(strategies, distribution);
    }
```

Figure 9.2: A snippet of the `flushSmartVault` function in spool-v2-core/DepositManager.sol#L188–L226

Lastly, the `_strategyRegistry.addDeposits` function is called with the computed distribution, which adds the amounts to deploy in the next `doHardWork` function call in the `_assetsDeposited` variable (figure 9.3).

```
   function addDeposits(address[] calldata strategies_, uint256[][] calldata
amounts)
        external
        onlyRole(ROLE_SMART_VAULT_MANAGER, msg.sender)
        returns (uint16a16)
    {
        uint16a16 indexes;
        for (uint256 i; i < strategies_.length; ++i) {
            address strategy = strategies_[i];

            uint256 latestIndex = _currentIndexes[strategy];
            indexes = indexes.set(i, latestIndex);

            for (uint256 j = 0; j < amounts[i].length; j++) {
                _assetsDeposited[strategy][latestIndex][j] += amounts[i][j];
            }
        }
```

```
        return indexes;
    }
```

Figure 9.3: The addDeposits function in spool-v2-core/StrategyRegistry.sol#L343–L361

The next time the doHardWork function is called, it will transfer the equivalent of the last deposit's amount instead of the sum of all deposits from the master wallet to the assigned strategy (figure 9.4).

```
   function doHardWork(DoHardWorkParameterBag calldata dhwParams) external
whenNotPaused {
                    ...
                    // Transfer deposited assets to the strategy.
                    for (uint256 k; k < assetGroup.length; ++k) {
                        if (_assetsDeposited[strategy][dhwIndex][k] > 0) {
                            _masterWallet.transfer(
                                IERC20(assetGroup[k]), strategy,
_assetsDeposited[strategy][dhwIndex][k]
                            );
                        }
                    }
                    ...
```

Figure 9.4: A snippet of the doHardWork function in spool-v2-core/StrategyRegistry.sol#L222–L341

Exploit Scenario

Bob deploys a smart vault. One hundred deposits are made before a smart vault flush is invoked, but only the last deposit's assets are deployed to the underlying strategies, severely impacting the smart vault's performance.

Recommendations

Short term, modify the depositAssets function so that it correctly tracks all deposits within a flush cycle, rather than just the last deposit.

Long term, improve the system's unit and integration tests: test a smart vault with a single strategy and multiple strategies to ensure that smart vaults behave correctly when funds are deposited and deployed to the underlying strategies.

10. Risk of malformed calldata of calls to guard contracts

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-10
Target: `managers/GuardManager.sol`	

Description

The `GuardManager` contract does not pad custom values while constructing the calldata for calls to guard contracts. The calldata could be malformed, causing the affected guard contract to give incorrect results or to always revert calls.

Guards for vaults are customizable checks that are executed on every user action. The result of a guard contract either approves or disapproves user actions. The `GuardManager` contract handles the logic to call guard contracts and to check their results (figure 10.1).

```
function runGuards(address smartVaultId, RequestContext calldata context) external
view {
        [...]
        bytes memory encoded = _encodeFunctionCall(smartVaultId, guard, context);
        (bool success, bytes memory data) =
guard.contractAddress.staticcall(encoded);
        _checkResult(success, data, guard.operator, guard.expectedValue, i);
    }
}
```

Figure 10.1: The `runGuards` function in `spool-v2-core/GuardManager.sol#L19–L33`

The arguments of the `runGuards` function include information related to the given user action and custom values defined at the time of guard definition. The `GuardManager.setGuards` function initializes the guards in the `GuardManager` contract.

Using the guard definition, the `GuardManager` contract manually constructs the calldata with the selected values from the user action information and the custom values (figure 10.2).

```
function _encodeFunctionCall(address smartVaultId, GuardDefinition memory guard,
RequestContext memory context)
    internal
    pure
    returns (bytes memory)
{
    [...]
    result = bytes.concat(result, methodID);
```

```
    for (uint256 i; i < paramsLength; ++i) {
        GuardParamType paramType = guard.methodParamTypes[i];

        if (paramType == GuardParamType.DynamicCustomValue) {
            result = bytes.concat(result, abi.encode(paramsEndLoc));
            paramsEndLoc += 32 + guard.methodParamValues[customValueIdx].length;
            customValueIdx++;
        } else if (paramType == GuardParamType.CustomValue) {
            result = bytes.concat(result, guard.methodParamValues[customValueIdx]);
            customValueIdx++;
        }
        [...]
    }

    customValueIdx = 0;
    for (uint256 i; i < paramsLength; ++i) {
        GuardParamType paramType = guard.methodParamTypes[i];

        if (paramType == GuardParamType.DynamicCustomValue) {
            result = bytes.concat(result,
abi.encode(guard.methodParamValues[customValueIdx].length / 32));
            result = bytes.concat(result, guard.methodParamValues[customValueIdx]);
            customValueIdx++;
        } else if (paramType == GuardParamType.CustomValue) {
            customValueIdx++;
        }
        [...]
    }

    return result;
}
```

Figure 10.2: The `_encodeFunctionCall` function in spool-v2-core/GuardManager.sol#L111–L177

However, the contract concatenates the custom values without considering their lengths and required padding. If these custom values are not properly padded at the time of guard initialization, the call will receive malformed data. As a result, either of the following could happen:

1. Every call to the guard contract will always fail, and user action transactions will always revert. The smart vault using the guard will become unusable.

2. The guard contract will receive incorrect arguments and return incorrect results. Invalid user actions could be approved, and valid user actions could be rejected.

Exploit Scenario

Bob deploys a smart vault and creates a guard for it. The guard contract takes only one custom value as an argument. Bob created the guard definition in `GuardManager` without

padding the custom value. Alice tries to deposit into the smart vault, and the guard contract is called for her action. The call to the guard contract fails, and the transaction reverts. The smart vault is unusable.

Recommendations

Short term, modify the associated code so that it verifies that custom values are properly padded before guard definitions are initialized in `GuardManager.setGuards`.

Long term, avoid implementing low-level manipulations. If such implementations are unavoidable, carefully review the Solidity documentation before implementing them to ensure that they are implemented correctly. Additionally, improve the user documentation with necessary technical details to properly use the system.

11. GuardManager does not account for all possible types when encoding guard arguments

Severity: **Low**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-SPL-11
Target: `managers/GuardManager.sol`	

Description

While encoding arguments for guard contracts, the `GuardManager` contract assumes that all static types are encoded to 32 bytes. This assumption does not hold for fixed-size static arrays and structs with only static type members. As a result, guard contracts could receive incorrect arguments, leading to unintended behavior.

The `GuardManager._encodeFunctionCall` function manually encodes arguments to call guard contracts (figure 11.1).

```
function _encodeFunctionCall(address smartVaultId, GuardDefinition memory guard,
RequestContext memory context)
    internal
    pure
    returns (bytes memory)
{
    bytes4 methodID = bytes4(keccak256(abi.encodePacked(guard.methodSignature)));
    uint256 paramsLength = guard.methodParamTypes.length;
    bytes memory result = new bytes(0);

    result = bytes.concat(result, methodID);
    uint16 customValueIdx = 0;
    uint256 paramsEndLoc = paramsLength * 32;

    // Loop through parameters and
    // - store values for simple types
    // - store param value location for dynamic types
    for (uint256 i; i < paramsLength; ++i) {
        GuardParamType paramType = guard.methodParamTypes[i];

        if (paramType == GuardParamType.DynamicCustomValue) {
            result = bytes.concat(result, abi.encode(paramsEndLoc));
            paramsEndLoc += 32 + guard.methodParamValues[customValueIdx].length;
            customValueIdx++;
        } else if (paramType == GuardParamType.CustomValue) {
            result = bytes.concat(result, guard.methodParamValues[customValueIdx]);
            customValueIdx++;
        }
```

```
        [...]
        } else if (paramType == GuardParamType.Assets) {
            result = bytes.concat(result, abi.encode(paramsEndLoc));
            paramsEndLoc += 32 + context.assets.length * 32;
        } else if (paramType == GuardParamType.Tokens) {
            result = bytes.concat(result, abi.encode(paramsEndLoc));
            paramsEndLoc += 32 + context.tokens.length * 32;
        } else {
            revert InvalidGuardParamType(uint256(paramType));
        }
    }
    [...]
    return result;
}
```

Figure 11.1: The `_encodeFunctionCall` function in spool-v2-core/GuardManager.sol#L111–L177

The function calculates the offset for dynamic type arguments assuming that every parameter, static or dynamic, takes exactly 32 bytes. However, fixed-length static type arrays and structs with only static type members are considered static. All static type values are encoded in-place, and static arrays and static structs could take more than 32 bytes.

As a result, the calculated offset for the start of dynamic type arguments could be wrong, which would cause incorrect values for these arguments to be set, resulting in unintended behavior. For example, the guard could approve invalid user actions and reject valid user actions or revert every call.

Exploit Scenario

Bob deploys a smart vault and creates a guard contract that takes the custom value of a fixed-length static array type. The guard contract uses `RequestContext` assets. Bob correctly creates the guard definition in `GuardManager`, but the `GuardManager._encodeFunctionCall` function incorrectly encodes the arguments. The guard contract fails to decode the arguments and always reverts the execution.

Recommendations

Short term, modify the `GuardManager._encodeFunctionCall` function so that it considers the encoding length of the individual parameters and calculates the offsets correctly.

Long term, avoid implementing low-level manipulations. If such implementations are unavoidable, carefully review the Solidity documentation before implementing them to ensure that they are implemented correctly.

12. Use of encoded values in guard contract comparisons could lead to opposite results

Severity: **Low**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-SPL-12
Target: `managers/GuardManager.sol`	

Description

The `GuardManager` contract compares the return value of a guard contract to an expected value. However, the contract uses encoded versions of these values in the comparison, which could lead to incorrect results for signed values with numerical comparison operators.

The `GuardManager` contract calls the guard contract and validates the return value using the `GuardManager._checkResult` function (figure 12.1).

```
function _checkResult(bool success, bytes memory returnValue, bytes2 operator,
bytes32 value, uint256 guardNum)
    internal
    pure
{
    if (!success) revert GuardError();

    bool result = true;

    if (operator == bytes2("==")) {
        result = abi.decode(returnValue, (bytes32)) == value;
    } else if (operator == bytes2("<=")) {
        result = abi.decode(returnValue, (bytes32)) <= value;
    } else if (operator == bytes2(">=")) {
        result = abi.decode(returnValue, (bytes32)) >= value;
    } else if (operator == bytes2("<")) {
        result = abi.decode(returnValue, (bytes32)) < value;
    } else if (operator == bytes2(">")) {
        result = abi.decode(returnValue, (bytes32)) > value;
    } else {
        result = abi.decode(returnValue, (bool));
    }

    if (!result) revert GuardFailed(guardNum);
}
```

Figure 12.1: The `_checkResult` function in spool-v2-core/GuardManager.sol#L80–L105

When a smart vault creator defines a guard using the `GuardManager.setGuards` function, they define a comparison operator and the expected value, which the `GuardManager` contract uses to compare with the return value of the guard contract.

The comparison is performed on the first 32 bytes of the ABI-encoded return value and the expected value, which will cause issues depending on the return value type.

First, the numerical comparison operators (<, >, <=, >=) are not well defined for `bytes32`; therefore, the contract treats encoded values with padding as `uint256` values before comparing them. This way of comparing values gives incorrect results for negative values of the `int<M>` type. The Solidity documentation includes the following description about the encoding of `int<M>` type values:

> *int<M>: enc(X) is the big-endian two's complement encoding of X, padded on the higher-order (left) side with 0xff bytes for negative X and with zero-bytes for non-negative X such that the length is 32 bytes.*

Figure 12.2: A description about the encoding of `int<M>` type values in the Solidity documentation

Because negative values are padded with `0xff` and positive values with `0x00`, the encoded negative values will be considered greater than the encoded positive values. As a result, the result of the comparison will be the opposite of the expected result.

Second, only the first 32 bytes of the return value are considered for comparison. This will lead to inaccurate results for return types that use more than 32 bytes to encode the value.

Exploit Scenario

Bob deploys a smart vault and intends to allow only users who own B NFTs to use it. B NFTs are implemented using ERC-1155. Bob uses the B contract as a guard with the comparison operator > and an expected value of `0`.

Bob calls the function `B.balanceOfBatch` to fetch the NFT balance of the user. `B.balanceOfBatch` returns `uint256[]`. The first 32 bytes of the return data contain the offset into the return data, which is always nonzero. The comparison passes for every user regardless of whether they own a B NFT. As a result, every user can use Bob's smart vault.

Recommendations

Short term, restrict the return value of a guard contract to a Boolean value. If that is not possible, document the limitations and risks surrounding the guard contracts. Additionally, consider manually checking new action guards with respect to these limitations.

Long term, avoid implementing low-level manipulations. If such implementations are unavoidable, carefully review the Solidity documentation before implementing them to ensure that they are implemented correctly.

13. Lack of contract existence checks on low-level calls

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-13
Target: `GuardManager.sol`, `Swapper.sol`	

Description

The `GuardManager` and `Swapper` contracts use low-level calls without contract existence checks. If the target address is incorrect or the contract at that address is destroyed, a low-level call will still return success.

The `Swapper.swap` function uses the `address().call(...)` function to swap tokens (figure 13.1).

```
function swap(
    address[] calldata tokensIn,
    SwapInfo[] calldata swapInfo,
    address[] calldata tokensOut,
    address receiver
) external returns (uint256[] memory tokenAmounts) {
    // Perform the swaps.
    for (uint256 i; i < swapInfo.length; ++i) {
        if (!exchangeAllowlist[swapInfo[i].swapTarget]) {
            revert ExchangeNotAllowed(swapInfo[i].swapTarget);
        }

        _approveMax(IERC20(swapInfo[i].token), swapInfo[i].swapTarget);

        (bool success, bytes memory data) =
swapInfo[i].swapTarget.call(swapInfo[i].swapCallData);
        if (!success) revert(SpoolUtils.getRevertMsg(data));
    }

    // Return unswapped tokens.
    for (uint256 i; i < tokensIn.length; ++i) {
        uint256 tokenInBalance = IERC20(tokensIn[i]).balanceOf(address(this));
        if (tokenInBalance > 0) {
            IERC20(tokensIn[i]).safeTransfer(receiver, tokenInBalance);
        }
    }
```

Figure 13.1: The swap function in `spool-v2-core/Swapper.sol#L29–L45`

The Solidity documentation includes the following warning:

The low-level functions call, delegatecall and staticcall return true as their first return value if the account called is non-existent, as part of the design of the EVM. Account existence must be checked prior to calling if needed.

Figure 13.2: The Solidity documentation details the necessity of executing existence checks before performing low-level calls.

Therefore, if the `swapTarget` address is incorrect or the target contract has been destroyed, the execution will not revert even if the swap is not successful.

We rated this finding as only a low-severity issue because the `Swapper` contract transfers the unswapped tokens to the receiver if a swap is not successful. However, the `CompoundV2Strategy` contract uses the `Swapper` contract to exchange COMP tokens for underlying tokens (figure 13.3).

```
function _compound(address[] calldata tokens, SwapInfo[] calldata swapInfo,
uint256[] calldata)
    internal
    override
    returns (int256 compoundedYieldPercentage)
{
    if (swapInfo.length > 0) {
        address[] memory markets = new address[](1);
        markets[0] = address(cToken);
        comptroller.claimComp(address(this), markets);

        uint256 compBalance = comp.balanceOf(address(this));

        if (compBalance > 0) {
            comp.safeTransfer(address(swapper), compBalance);
            address[] memory tokensIn = new address[](1);
            tokensIn[0] = address(comp);
            uint256 swappedAmount = swapper.swap(tokensIn, swapInfo, tokens,
address(this))[0];

            if (swappedAmount > 0) {
                uint256 cTokenBalanceBefore = cToken.balanceOf(address(this));
                _depositToCompoundProtocol(IERC20(tokens[0]), swappedAmount);
                uint256 cTokenAmountCompounded = cToken.balanceOf(address(this)) -
cTokenBalanceBefore;

                compoundedYieldPercentage =
_calculateYieldPercentage(cTokenBalanceBefore, cTokenAmountCompounded);
            }
        }
    }
}
```

Figure 13.3: The `_compound` function in `spool-v2-core/CompoundV2Strategy.sol`

If the swap operation fails, the COMP will stay in `CompoundV2Strategy`. This will cause users to lose the yield they would have gotten from compounding. Because the swap operation fails silently, the "do hard worker" may not notice that yield is not compounding. As a result, users will receive less in profit than they otherwise would have.

The `GuardManager.runGuards` function, which uses the `address().staticcall()` function, is also affected by this issue. However, the return value of the call is decoded, so the calls would not fail silently.

Exploit Scenario

The Spool team deploys `CompoundV2Strategy` with a market that gives COMP tokens to its users. While executing the `doHardWork` function for smart vaults using `CompoundV2Strategy`, the "do hard worker" sets the `swapTarget` address to an incorrect address. The swap operation to exchange COMP to the underlying token fails silently. The gained yield is not deposited into the market. The users receive less in profit.

Recommendations

Short term, implement a contract existence check before the low-level calls in `GuardManager.runGuards` and `Swapper.swap`.

Long term, avoid implementing low-level calls. If such calls are unavoidable, carefully review the Solidity documentation, particularly the "Warnings" section, before implementing them to ensure that they are implemented correctly.

14. Incorrect use of exchangeRates in doHardWork

Severity: **High**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-14
Target: `managers/StrategyRegistry.sol`	

Description

The `StrategyRegistry` contract's `doHardWork` function fetches the `exchangeRates` for all of the tokens involved in the "do hard work" process, and then it iterates over the strategies and saves the `exchangeRates` values for the current strategy's tokens in the `assetGroupExchangeRates` variable; however, when `doHardWork` is called for a strategy, the `exchangeRates` variable rather than the `assetGroupExchangeRates` variable is passed, resulting in the use of incorrect exchange rates.

```
   function doHardWork(DoHardWorkParameterBag calldata dhwParams) external
whenNotPaused {
           ...
           // Get exchange rates for tokens and validate them against slippages.
            uint256[] memory exchangeRates =
SpoolUtils.getExchangeRates(dhwParams.tokens, _priceFeedManager);
            for (uint256 i; i < dhwParams.tokens.length; ++i) {
                if (
                    exchangeRates[i] < dhwParams.exchangeRateSlippages[i][0]
                        || exchangeRates[i] > dhwParams.exchangeRateSlippages[i][1]
                ) {
                    revert ExchangeRateOutOfSlippages();
                }
            }
           ...

       // Get exchange rates for this group of strategies.
       uint256 assetGroupId = IStrategy(dhwParams.strategies[i][0]).assetGroupId();
       address[] memory assetGroup = IStrategy(dhwParams.strategies[i][0]).assets();
       uint256[] memory assetGroupExchangeRates = new uint256[](assetGroup.length);

           for (uint256 j; j < assetGroup.length; ++j) {
                bool found = false;

                for (uint256 k; k < dhwParams.tokens.length; ++k) {
                     if (assetGroup[j] == dhwParams.tokens[k]) {
                        assetGroupExchangeRates[j] = exchangeRates[k];

                        found = true;
                        break;
```

```
                }
            }
...
            // Do the hard work on the strategy.
             DhwInfo memory dhwInfo = IStrategy(strategy).doHardWork(
                 StrategyDhwParameterBag({
                     swapInfo: dhwParams.swapInfo[i][j],
                     compoundSwapInfo: dhwParams.compoundSwapInfo[i][j],
                     slippages: dhwParams.strategySlippages[i][j],
                     assetGroup: assetGroup,
                     exchangeRates: exchangeRates,
                     withdrawnShares: _sharesRedeemed[strategy][dhwIndex],
                     masterWallet: address(_masterWallet),
                     priceFeedManager: _priceFeedManager,
                     baseYield: dhwParams.baseYields[i][j],
                     platformFees: platformFeesMemory
                 })
             );
            // Bookkeeping.
            _dhwAssetRatios[strategy] = IStrategy(strategy).assetRatio();
            _exchangeRates[strategy][dhwIndex].setValues(exchangeRates);
            ...
```

Figure 14.1: A snippet of the doHardWork function in spool-v2-core/StrategyRegistry.sol#L222–L341

The `exchangeRates` values are used by a strategy's `doHardWork` function to calculate how many assets in USD value are to be deposited and how many in USD value are currently deposited in the strategy. As a consequence of using `exchangeRates` rather than `assetGroupExchangeRates`, the contract will return incorrect values.

Additionally, the `_exchangeRates` variable is returned by the `strategyAtIndexBatch` function, which is used when simulating deposits.

Exploit Scenario

Bob deploys a smart vault, and users start depositing into it. However, the first time `doHardWork` is called, they notice that the deposited assets and the reported USD value deposited into the strategies are incorrect. They panic and start withdrawing all of the funds.

Recommendations

Short term, replace `exchangeRates` with `assetGroupExchangeRates` in the relevant areas of `doHardWork` and where it sets the `_exchangeRates` variable.

Long term, improve the system's unit and integration tests to verify that the deposited value in a strategy is the expected amount. Additionally, when reviewing the code, look for local variables that are set but then never used; this is a warning sign that problems may arise.

15. LinearAllocationProvider could return an incorrect result

Severity: **Medium**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-SPL-15
Target: `providers/LinearAllocationProvider.sol`	

Description

The `LinearAllocationProvider` contract returns an incorrect result when the given smart vault has a `riskTolerance` value of `-8` due to an incorrect literal value in the `riskArray` variable.

```
   function calculateAllocation(AllocationCalculationInput calldata data) external
pure returns (uint256[] memory) {
       ...
       uint24[21] memory riskArray = [
            100000,
            95000,
            900000,
            ...
        ];
        ...
        uint8 riskt = uint8(data.riskTolerance + 10); // from 0 to 20

        for (uint8 i; i < data.apys.length; ++i) {
            ...
            results[i] = apy * riskArray[uint8(20 - riskt)] + risk *
riskArray[uint8(riskt)];

            resSum += results[i];
        }

        uint256 resSum2;
        for (uint8 i; i < results.length; ++i) {
            results[i] = FULL_PERCENT * results[i] / resSum;
            resSum2 += results[i];
        }

        results[0] += FULL_PERCENT - resSum2;

        return results;
```

Figure 15.1: A snippet of the `calculateAllocation` function in spool-v2-core/LinearAllocationProvider.sol#L9–L67

The `riskArray`'s third element is incorrect; this affects the computed allocation for smart vaults that have a `riskTolerance` value of `-8` because the `riskt` variable would be 2, which is later used as index for the `riskArray`.

The subexpression `risk * riskArray[uint8(rikst)]` is incorrect by a factor of 10.

Exploit Scenario

Bob deploys a smart vault with a `riskTolerance` value of `-8` and an empty `strategyAllocation` value. The allocation between the strategies is computed on the spot using the `LinearAllocationProvider` contract, but the allocation is wrong.

Recommendations

Short term, replace `900000` with `90000` in the `calculateAllocation` function.

Long term, improve the system's unit and integration tests to catch issues such as this. Document the use and meaning of constants such as the values in `riskArray`. This will make it more likely that the Spool team will find these types of mistakes.

16. Incorrect formula used for adding/subtracting two yields

Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-16
Target: `manager/StrategyRegistry.sol`, `strategies/Strategy.sol`	

Description

The `doHardWork` function adds two yields with different base values to compute the given strategy's total yield, which results in the collection of fewer ecosystem fees and treasury fees.

It is incorrect to add two yields that have different base values. The correct formula to compute the total yield from two consecutive yields `Y1` and `Y2` is `Y1 + Y2 + (Y1*Y2)`.

The `doHardWork` function in the `Strategy` contract adds the protocol yield and the rewards yield to calculate the given strategy's total yield. The protocol yield percentage is calculated with the base value of the strategy's total assets at the start of the current "do hard work" cycle, while the rewards yield percentage is calculated with the base value of the total assets currently owned by the strategy.

```
dhwInfo.yieldPercentage = _getYieldPercentage(dhwParams.baseYield);
dhwInfo.yieldPercentage += _compound(dhwParams.assetGroup,
dhwParams.compoundSwapInfo, dhwParams.slippages);
```

Figure 16.1: A snippet of the `doHardWork` function in `spool-v2-core/Strategy.sol#L95–L96`

Therefore, the total yield of the strategy is computed as less than its actual yield, and the use of this value to compute fees results in the collection of fewer fees for the platform's governance system.

Same issue also affects the computation of the total yield of a strategy on every "do hard work" cycle:

```
_stateAtDhw[strategy][dhwIndex] = StateAtDhwIndex({
    sharesMinted: uint128(dhwInfo.sharesMinted),
    totalStrategyValue: uint128(dhwInfo.valueAtDhw),
    totalSSTs: uint128(dhwInfo.totalSstsAtDhw),
    yield: int96(dhwInfo.yieldPercentage) + _stateAtDhw[strategy][dhwIndex -
1].yield, // accumulate the yield from before
    timestamp: uint32(block.timestamp)
```

```
});
```

Figure 16.2: A snippet of the doHardWork function in spool-v2-core/StrategyRegistry.sol#L331–L337

This value of the total yield of a strategy is used to calculate the management fees for a given smart vault, which results in fewer fees paid to the smart vault owner.

Exploit Scenario

The Spool team deploys the system. Alice deposits 1,000 tokens into a vault, which mints 1,000 strategy share tokens for the vault. On the next "do hard work" execution, the tokens earn 8% yield and 30 reward tokens from the protocol. The 30 reward tokens are then exchanged for 20 deposit tokens. At this point, the total tokens earned by the strategy are 100 and the total yield is 10%. However, the `doHardWork` function computes the total yield as 9.85%, which is incorrect, resulting in fewer fees collected for the platform.

Recommendations

Short term, use the correct formula to calculate a given strategy's total yield in both the `Strategy` contract and the `StrategyRegistry` contract.

Note that the `syncDepositsSimulate` function subtracts a strategy's total yield at different "do hard work" indexes in `DepositManager.sol#L322–L326` to compute the difference between the strategy's yields between two "do hard work" cycles. After fixing this issue, this function's computation will be incorrect.

Long term, review the entire codebase to find all of the mathematical formulas used. Document these formulas, their assumptions, and their derivations to avoid the use of incorrect formulas.

17. Smart vaults with re-registered strategies will not be usable

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-SPL-17
Target: `manager/StrategyRegistry.sol`	

Description

The `StrategyRegistry` contract does not clear the state related to a strategy when removing it. As a result, if the removed strategy is registered again, the `StrategyRegistry` contract will still contain the strategy's previous state, resulting in a temporary DoS of the smart vaults using it.

The `StrategyRegistry.registerStrategy` function is used to register a strategy and to initialize the state related to it (figure 17.1). `StrategyRegistry` tracks the state of the strategies by using their address.

```
function registerStrategy(address strategy) external {
    _checkRole(ROLE_SPOOL_ADMIN, msg.sender);
    if (_accessControl.hasRole(ROLE_STRATEGY, strategy)) revert
StrategyAlreadyRegistered({address_: strategy});

    _accessControl.grantRole(ROLE_STRATEGY, strategy);
    _currentIndexes[strategy] = 1;
    _dhwAssetRatios[strategy] = IStrategy(strategy).assetRatio();
    _stateAtDhw[address(strategy)][0].timestamp = uint32(block.timestamp);
}
```

Figure 17.1: The `registerStrategy` function in `spool-v2-core/StrategyRegistry.sol`

The `StrategyRegistry._removeStrategy` function is used to remove a strategy by revoking its `ROLE_STRATEGY` role.

```
function _removeStrategy(address strategy) private {
    if (!_accessControl.hasRole(ROLE_STRATEGY, strategy)) revert
InvalidStrategy({address_: strategy});
    _accessControl.revokeRole(ROLE_STRATEGY, strategy);
}
```

Figure 17.2: The `_removeStrategy` function in `spool-v2-core/StrategyRegistry.sol`

While removing a strategy, `StrategyRegistry` contract does not remove the state related to that strategy. As a result, when that strategy is registered again, `StrategyRegistry` will contain values from the previous period. This could make the smart vaults using the

strategy unusable or cause the unintended transfer of assets between other strategies and this strategy.

Exploit Scenario

Strategy S is registered. `StrategyRegistry._currentIndex[S]` is equal to 1. Alice creates a smart vault X that uses strategy S. Bob deposits 1 million WETH into smart vault X. `StrategyRegistry._assetsDeposited[S][1][WETH]` is equal to 1 million WETH. The `doHardWork` function is called for strategy S. WETH is transferred from the master wallet to strategy S and is deposited into the protocol.

A Spool system admin removes strategy S upon hearing that the protocol is being exploited. However, the admin realizes that the protocol is not being exploited and re-registers strategy S. `StrategyRegistry._currentIndex[S]` is set to 1. `StrategyRegistry._assetsDeposited[S][1][WETH]` is not set to zero and is still equal to 1 million WETH.

Alice creates a new vault with strategy S. When `doHardWork` is called for strategy S, `StrategyRegistry` tries to transfer 1 million WETH to the strategy. The master wallet does not have those assets, so `doHardWork` fails for strategy S. The smart vault becomes unusable.

Recommendations

Short term, modify the `StrategyRegistry._removeStrategy` function so that it clears states related to removed strategies if re-registering strategies is an intended use case. If this is not an intended use case, modify the `StrategyRegistry.registerStrategy` function so that it verifies that newly registered strategies have not been previously registered.

Long term, properly document all intended use cases of the system and implement comprehensive tests to ensure that the system behaves as expected.

18. Incorrect handling of partially burned NFTs results in incorrect SVT balance calculation

Severity: **Low**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-SPL-18
Target: `manager/SmartVaultManager.sol`, `SmartVault.sol`	

Description

The `SmartVault._afterTokenTransfer` function removes the given NFT ID from the `SmartVault._activeUserNFTIds` array even if only a fraction of it is burned. As a result, the `SmartVaultManager.getUserSVTBalance` function, which uses `SmartVault._activeUserNFTIds`, will show less than the given user's actual balance.

`SmartVault._afterTokenTransfer` is executed after every token transfer (figure 18.1).

```
function _afterTokenTransfer(
    address,
    address from,
    address to,
    uint256[] memory ids,
    uint256[] memory,
    bytes memory
) internal override {
    // burn
    if (to == address(0)) {
        uint256 count = _activeUserNFTCount[from];
        for (uint256 i; i < ids.length; ++i) {
            for (uint256 j = 0; j < count; j++) {
                if (_activeUserNFTIds[from][j] == ids[i]) {
                    _activeUserNFTIds[from][j] = _activeUserNFTIds[from][count - 1];
                    count--;
                    break;
                }
            }
        }

        _activeUserNFTCount[from] = count;
        return;
    }
[...]
}
```

Figure 18.1: A snippet of the `_afterTokenTransfer` function in `spool-v2-core/SmartVault.sol`

It removes the burned NFT from `_activeUserNFTIds`. However, it does not consider the amount of the NFT that was burned. As a result, NFTs that are not completely burned will not be considered active by the vault.

`SmartVaultManager.getUserSVTBalance` uses `SmartVault._activeUserNFTIds` to calculate a given user's SVT balance (figure 18.2).

```
function getUserSVTBalance(address smartVaultAddress, address userAddress) external
view returns (uint256) {
    if (_accessControl.smartVaultOwner(smartVaultAddress) == userAddress) {
        (, uint256 ownerSVTs,, uint256 fees) = _simulateSync(smartVaultAddress);
        return ownerSVTs + fees;
    }

    uint256 currentBalance = ISmartVault(smartVaultAddress).balanceOf(userAddress);
    uint256[] memory nftIds =
ISmartVault(smartVaultAddress).activeUserNFTIds(userAddress);

    if (nftIds.length > 0) {
        currentBalance += _simulateNFTBurn(smartVaultAddress, userAddress, nftIds);
    }

    return currentBalance;
}
```

Figure 18.2: The `getUserSVTBalance` function in `spool-v2-core/SmartVaultManager.sol`

Because partial NFTs are not present in `SmartVault._activeUserNFTIds`, the calculated balance will be less than the user's actual balance. The front end using `getUserSVTBalance` will show incorrect balances to users.

Exploit Scenario

Alice deposits assets into a smart vault and receives a D-NFT. Alice's assets are deposited to the protocols after `doHardWork` is called. Alice claims SVTs by burning a fraction of her D-NFT. The smart vault removes the D-NFT from `_activeUserNFTIds`. Alice checks her SVT balance and panics when she sees less than what she expected. She withdraws all of her assets from the system.

Recommendations

Short term, add a check to the `_afterTokenTransfer` function so that it checks the balance of the NFT that is burned and removes the NFT from `_activeUserNFTIds` only when the NFT is burned completely.

Long term, improve the system's unit and integration tests to extensively test view functions.

19. Transfers of D-NFTs result in double counting of SVT balance

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-SPL-19
Target: `manager/SmartVaultManager.sol`, `SmartVault.sol`	

Description

The `_activeUserNFTIds` and `_activeUserNFTCount` variables are not updated for the sender account on the transfer of NFTs. As a result, SVTs for transferred NFTs will be counted twice, causing the system to show an incorrect SVT balance.

The `_afterTokenTransfer` hook in the `SmartVault` contract is executed after every token transfer to update information about users' active NFTs:

```
function _afterTokenTransfer(
    address,
    address from,
    address to,
    uint256[] memory ids,
    uint256[] memory,
    bytes memory
) internal override {
    // burn
    if (to == address(0)) {
        ...
        return;
    }
    // mint or transfer
    for (uint256 i; i < ids.length; ++i) {
        _activeUserNFTIds[to][_activeUserNFTCount[to]] = ids[i];
        _activeUserNFTCount[to]++;
    }

}
```

Figure 19.1: A snippet of the `_afterTokenTransfer` function in `spool-v2-core/SmartVault.sol`

When a user transfers an NFT to another user, the function adds the NFT ID to the active NFT IDs of the receiver's account but does not remove the ID from the active NFT IDs of the sender's account. Additionally, the active NFT count is not updated for the sender's account.

The `getUserSVTBalance` function of the `SmartVaultManager` contract uses the `SmartVault` contract's `_activeUserNFTIds` array to calculate a given user's SVT balance:

```
function getUserSVTBalance(address smartVaultAddress, address userAddress) external
view returns (uint256) {
    if (_accessControl.smartVaultOwner(smartVaultAddress) == userAddress) {
        (, uint256 ownerSVTs,, uint256 fees) = _simulateSync(smartVaultAddress);
        return ownerSVTs + fees;
    }

    uint256 currentBalance = ISmartVault(smartVaultAddress).balanceOf(userAddress);
    uint256[] memory nftIds =
ISmartVault(smartVaultAddress).activeUserNFTIds(userAddress);

    if (nftIds.length > 0) {
        currentBalance += _simulateNFTBurn(smartVaultAddress, userAddress, nftIds);
    }

    return currentBalance;
}
```

Figure 19.2: The `getUserSVTBalance` function in `spool-v2-core/SmartVaultManager.sol`

Because transferred NFT IDs are active for both senders and receivers, the SVTs corresponding to the NFT IDs will be counted for both users. This double counting will keep increasing the SVT balance for users with every transfer, causing an incorrect balance to be shown to users and third-party integrators.

Exploit Scenario

Alice deposits assets into a smart vault and receives a D-NFT. Alice's assets are deposited into the protocols after `doHardWork` is called. Alice transfers the D-NFT to herself. The `SmartVault` contract adds the D-NFT ID to `_activeUserNFTIds` for Alice again. Alice checks her SVT balance and sees double the balance she had before.

Recommendations

Short term, modify the `_afterTokenTransfer` function so that it removes NFT IDs from the active NFT IDs for the sender's account when users transfer D-NFTs and W-NFTs.

Long term, add unit test cases for all possible user interactions to catch issues such as this.

20. Flawed loop for syncing flushes results in higher management fees

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-SPL-20
Target: `manager/SmartVaultManager.sol`, `manager/DepositManager.sol`	

Description

The loop used to sync flush indexes in the `SmartVaultManager` contract computes an inflated value of the `oldTotalSVTs` variable, which results in higher management fees paid to the smart vault owner.

The `_syncSmartVault` function in the `SmartVaultManager` contract implements a loop to process every flush index from `flushIndex.toSync` to `flushIndex.current`:

```
while (flushIndex.toSync < flushIndex.current) {
    ...

    DepositSyncResult memory syncResult = _depositManager.syncDeposits(
        smartVault,
        [flushIndex.toSync, bag.lastDhwSynced, bag.oldTotalSVTs],
        strategies_,
        [indexes, _getPreviousDhwIndexes(smartVault, flushIndex.toSync)],
        tokens,
        bag.fees
    );

    bag.newSVTs += syncResult.mintedSVTs;
    bag.feeSVTs += syncResult.feeSVTs;
    bag.oldTotalSVTs += bag.newSVTs;
    bag.lastDhwSynced = syncResult.dhwTimestamp;

    emit SmartVaultSynced(smartVault, flushIndex.toSync);
    flushIndex.toSync++;
}
```

Figure 20.1: A snippet of the `_syncSmartVault` function in `spool-v2-core/SmartVaultManager.sol`

This loop adds the value of `mintedSVTs` to the `newSVTs` variables and then computes the value of `oldTotalSVTs` by adding `newSVTs` to it in every iteration. Because `mintedSVTs` are added in every iteration, new minted SVTs are added for each flush index multiple times when the loop is iterated more than once.

The value of `oldTotalSVTs` is then passed to the `syncDeposit` function of the `DepositManager` contract, which uses it to compute the management fee for the smart vault. The use of the inflated value of `oldTotalSVTs` causes higher management fees to be paid to the smart vault owner.

Exploit Scenario

Alice deposits assets into a smart vault and flushes it. Before `doHardWork` is executed, Bob deposits assets into the same smart vault and flushes it. At this point, `flushIndex.current` has been increased twice for the smart vault. After the execution of `doHardWork`, the loop to sync the smart vault is iterated twice. As a result, a double management fee is paid to the smart vault owner, and Alice and Bob lose assets.

Recommendations

Short term, modify the loop so that `syncResult.mintedSVTs` is added to `bag.oldTotalSVTs` instead of `bag.newSVTs`.

Long term, be careful when implementing accumulators in loops. Add test cases for multiple interactions to catch such issues.

21. Incorrect ghost strategy check

Severity: **Informational**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-SPL-21
Target: `manager/StrategyRegistry.sol`	

Description

The `emergencyWithdraw` and `redeemStrategyShares` functions incorrectly check whether a strategy is a ghost strategy after checking that the strategy has a `ROLE_STRATEGY` role.

```
function emergencyWithdraw(
     address[] calldata strategies,
     uint256[][] calldata withdrawalSlippages,
     bool removeStrategies
 ) external onlyRole(ROLE_EMERGENCY_WITHDRAWAL_EXECUTOR, msg.sender) {
     for (uint256 i; i < strategies.length; ++i) {
         _checkRole(ROLE_STRATEGY, strategies[i]);
         if (strategies[i] == _ghostStrategy) {
             continue;
         }
     [...]
```

Figure 21.1: A snippet the `emergencyWithdraw` function
spool-v2-core/StrategyRegistry.sol#L456–L465

```
function redeemStrategyShares(
     address[] calldata strategies,
     uint256[] calldata shares,
     uint256[][] calldata withdrawalSlippages
 ) external {
     for (uint256 i; i < strategies.length; ++i) {
         _checkRole(ROLE_STRATEGY, strategies[i]);
         if (strategies[i] == _ghostStrategy) {
             continue;
         }
     [...]
```

Figure 21.2: A snippet the `emergencyWithdraw` function
spool-v2-core/StrategyRegistry.sol#L477–L486

A ghost strategy will never have the `ROLE_STRATEGY` role, so both functions will always incorrectly revert if a ghost strategy is passed in the strategies array.

Exploit Scenario
Bob calls `redeemStrategyShares` with the ghost strategy in `strategies` and the transaction unexpectedly reverts.

Recommendations
Short term, modify the affected functions so that they verify whether the given strategy is a ghost strategy before checking the role with `_checkRole`.

Long term, clearly document which roles a contract should have and implement the appropriate checks to verify them.

22. Reward configuration not initialized properly when reward is zero

Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-22
Target: `rewards/RewardManager.sol`	

Description

The `RewardManager.addToken` function, which adds a new reward token for the given smart vault, does not initialize all configuration variables when the initial reward is zero. As a result, all calls to the `RewardManager.extendRewardEmission` function will fail, and rewards cannot be added for that vault.

`RewardManager.addToken` adds a new reward token for the given smart vault. The reward tokens for a smart vault are tracked using the `RewardManager.rewardConfiguration` function. The `tokenAdded` value of the configuration is used to check whether the token has already been added for the vault (figure 22.1).

```
function addToken(address smartVault, IERC20 token, uint32 rewardsDuration, uint256
reward)
    external
    onlyAdminOrVaultAdmin(smartVault, msg.sender)
    exceptUnderlying(smartVault, token)
{
    RewardConfiguration storage config = rewardConfiguration[smartVault][token];

    if (tokenBlacklist[smartVault][token]) revert
RewardTokenBlacklisted(address(token));
    if (config.tokenAdded != 0) revert RewardTokenAlreadyAdded(address(token));
    if (rewardsDuration == 0) revert InvalidRewardDuration();
    if (rewardTokensCount[smartVault] > 5) revert RewardTokenCapReached();

    rewardTokens[smartVault][rewardTokensCount[smartVault]] = token;
    rewardTokensCount[smartVault]++;

    config.rewardsDuration = rewardsDuration;

    if (reward > 0) {
        _extendRewardEmission(smartVault, token, reward);
    }
}
```

Figure 22.1: The addToken function in spool-v2-core/RewardManager.sol#L81–L101

However, `RewardManager.addToken` does not update `config.tokenAdded`, and the `_extendRewardEmission` function, which updates `config.tokenAdded`, is called only when the reward is greater than zero.

`RewardManager.extendRewardEmission` is the only entry point to add rewards for a vault. It checks whether token has been previously added by verifying that `tokenAdded` is greater than zero (figure 22.2).

```
function extendRewardEmission(address smartVault, IERC20 token, uint256 reward,
uint32 rewardsDuration)
    external
    onlyAdminOrVaultAdmin(smartVault, msg.sender)
    exceptUnderlying(smartVault, token)
{
    if (tokenBlacklist[smartVault][token]) revert
RewardTokenBlacklisted(address(token));
    if (rewardsDuration == 0) revert InvalidRewardDuration();
    if (rewardConfiguration[smartVault][token].tokenAdded == 0) {
        revert InvalidRewardToken(address(token));
    }
    [...]
}
```

Figure 22.2: The `extendRewardEmission` function in spool-v2-core/RewardManager.sol#L106–L119

Because `tokenAdded` is not initialized when the initial rewards are zero, the vault admin cannot add the rewards for the vault in that token.

The impact of this issue is lower because the vault admin can use the `RewardManager.removeReward` function to remove the token and add it again with a nonzero initial reward. Note that the vault admin can only remove the token without blacklisting it because the `config.periodFinish` value is also not initialized when the initial reward is zero.

Exploit Scenario

Alice is the admin of a smart vault. She adds a reward token for her smart vault with the initial reward set to zero. Alice tries to add rewards using `extendRewardEmission`, and the transaction fails. She cannot add rewards for her smart vault. She has to remove the token and re-add it with a nonzero initial reward.

Recommendations

Short term, use a separate Boolean variable to track whether a token has been added for a smart vault, and have `RewardManager.addToken` initialize that variable.

Long term, improve the system's unit tests to cover all execution paths.

23. Missing function for removing reward tokens from the blacklist

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-SPL-23
Target: `rewards/RewardManager.sol`	

Description

A Spool admin can blacklist a reward token for a smart vault through the `RewardManager` contract, but they cannot remove it from the blacklist. As a result, a reward token cannot be used again once it is blacklisted.

The `RewardManager.forceRemoveReward` function blacklists the given reward token by updating the `RewardManager.tokenBlacklist` array (figure 23.1). Blacklisted tokens cannot be used as rewards.

```
function forceRemoveReward(address smartVault, IERC20 token) external
onlyRole(ROLE_SPOOL_ADMIN, msg.sender) {
    tokenBlacklist[smartVault][token] = true;
    _removeReward(smartVault, token);

    delete rewardConfiguration[smartVault][token];
}
```

Figure 23.1: The `forceRemoveReward` function in spool-v2-core/RewardManager.sol#L160–L165

However, `RewardManager` does not have a function to remove tokens from the blacklist. As a result, if the Spool admin accidentally blacklists a token, then the smart vault admin will never be able to use that token to send rewards.

Exploit Scenario

Alice is the admin of a smart vault. She adds WETH and token A as rewards. The value of token A declines rapidly, so a Spool admin decides to blacklist the token for Alice's vault. The Spool admin accidentally supplies the WETH address in the call to `forceRemoveReward`. As a result, WETH is blacklisted, and Alice cannot send rewards in WETH.

Recommendations

Short term, add a function with the proper access controls to remove tokens from the blacklist.

Long term, improve the system's unit tests to cover all execution paths.

24. Risk of unclaimed shares due to loss of precision in reallocation operations

Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-24
Target: `libraries/ReallocationLib.sol`	

Description

The `ReallocationLib.calculateReallocation` function releases strategy shares and calculates their USD value. The USD value is later converted into strategy shares in the `ReallocationLib.doReallocation` function. Because the conversion operations always round down, the number of shares calculated in `doReallocation` will be less than the shares released in `calculateReallocation`. As a result, some shares released in `calculateReallocation` will be unclaimed, as `ReallocationLib` distributes only the shares computed in `doReallocation`.

`ReallocationLib.calculateAllocation` calculates the USD value that needs to be withdrawn from each of the strategies used by smart vaults (figure 24.1). The smart vaults release the shares equivalent to the calculated USD value.

```
/**
 * @dev Calculates reallocation needed per smart vault.
[...]
 * @return Reallocation of the smart vault:
 * - first index is 0 or 1
 * - 0:
 *   - second index runs over smart vault's strategies
 *   - value is USD value that needs to be withdrawn from the strategy
[...]
 */
function calculateReallocation(
[...]
) private returns (uint256[][] memory) {
    [...]
        } else if (targetValue < currentValue) {
            // This strategy needs withdrawal.
            [...]
            IStrategy(smartVaultStrategies[i]).releaseShares(smartVault,
sharesToRedeem);

            // Recalculate value to withdraw based on released shares.
            reallocation[0][i] = IStrategy(smartVaultStrategies[i]).totalUsdValue()
* sharesToRedeem
```

```
                / IStrategy(smartVaultStrategies[i]).totalSupply();
            }
        }

        return reallocation;
    }
```

Figure 24.1: The `calculateReallocation` function in spool-v2-core/ReallocationLib.sol#L161–L207

The `ReallocationLib.buildReallocationTable` function calculates the `reallocationTable` value. The `reallocationTable[i][j][0]` value represents the USD amount that should move from strategy `i` to strategy `j` (figure 24.2). These USD amounts are calculated using the USD values of the released shares computed in `ReallocationLib.calculateReallocation` (represented by `reallocation[0][i]` in figure 24.1).

```
/**
 [...]
 * @return Reallocation table:
 * - first index runs over all strategies i
 * - second index runs over all strategies j
 * - third index is 0, 1 or 2
 *   - 0: value represent USD value that should be withdrawn by strategy i and
deposited into strategy j
 */
function buildReallocationTable(
    [...]
) private pure returns (uint256[][][] memory) {
```

Figure 24.2: A snippet of `buildReallocationTable` function in spool-v2-core/ReallocationLib.sol#L209–L228

`ReallocationLib.doReallocation` calculates the total USD amount that should be withdrawn from a strategy (figure 24.3). This total USD amount is exactly equal to the sum of the USD values needed to be withdrawn from the strategy for each of the smart vaults. The `doReallocation` function converts the total USD value to the equivalent number of strategy shares. The `ReallocationLib` library withdraws this exact number of shares from the strategy and distributes them to other strategies that require deposits of these shares.

```
function doReallocation(
    [...]
    uint256[][][] memory reallocationTable
) private {

    // Distribute matched shares and withdraw unamatched ones.
    for (uint256 i; i < strategies.length; ++i) {
```

```
        [...]
        {
            uint256[2] memory totals;
            // totals[0] -> total withdrawals

            for (uint256 j; j < strategies.length; ++j) {
                totals[0] += reallocationTable[i][j][0];
                [...]
            }
            // Calculate amount of shares to redeem and to distribute.
            uint256 sharesToDistribute = // first store here total amount of shares
that should have been withdrawn
             IStrategy(strategies[i]).totalSupply() * totals[0] /
IStrategy(strategies[i]).totalUsdValue();
             [...]
        }
[...]
```

Figure 24.3: A snippet of the `doReallocation` function in spool-v2-core/ReallocationLib.sol#L285–L350

Theoretically, the shares calculated for a strategy should be equal to the shares released by all of the smart vaults for that strategy. However, there is a loss of precision in both the `calculateReallocation` function's calculation of the USD value of released shares and the `doReallocation` function's conversion of the combined USD value to strategy shares. As a result, the number of shares released by all of the smart vaults will be less than the shares calculated in `calculateReallocation`. Because the `ReallocationLib` library only distributes these calculated shares, there will be some unclaimed strategy shares as dust.

It is important to note that the rounding error could be greater than one in the context of multiple smart vaults. Additionally, the error could be even greater if the conversion results were rounded in the opposite direction: in that case, if the calculated shares were greater than the released shares, the reallocation would fail when burn and claim operations are executed.

Recommendations

Short term, modify the code so that it stores the number of shares released in `calculateReallocation`, and implement dustless calculations to build the `reallocationTable` value with the share amounts and the USD amounts. Have `doReallocation` use this `reallocationTable` value to calculate the value of `sharesToDistribute`.

Long term, use Echidna to test system and mathematical invariants.

25. Curve3CoinPoolAdapter's _addLiquidity reverts due to incorrect amounts deposited

Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-25
Target: `strategies/curve/CurveAdapter.sol`	

Description

The `_addLiquidity` function loops through the `amounts` array but uses an additional element to keep track of whether deposits need to be made in the `Strategy.doHardWork` function. As a result, `_addLiquidity` overwrites the number of tokens to send for the first asset, causing far fewer tokens to be deposited than expected, thus causing the transaction to revert due to the slippage check.

```
    function _addLiquidity(uint256[] memory amounts, uint256 slippage) internal {
        uint256[N_COINS] memory curveAmounts;

        for (uint256 i; i < amounts.length; ++i) {
            curveAmounts[assetMapping().get(i)] = amounts[i];
        }

        ICurve3CoinPool(pool()).add_liquidity(curveAmounts, slippage);
    }
```

Figure 25.1: The `_addLiquidity` function in spool-v2-core/CurveAdapter.sol#L12–L20

The last element in the `doHardWork` function's `assetsToDeposit` array keeps track of the deposits to be made and is incremented by one on each iteration of assets in `assetGroup` if that asset has tokens to deposit. This variable is then passed to the `_depositToProtocol` function and then, for strategies that use the `Curve3CoinPoolAdapter`, is passed to `_addLiquidity` in the `amounts` parameter. When `_addLiquidity` iterates over the last element in the `amounts` array, the `assetMapping().get(i)` function will return `0` because `i` in `assetMapping` is uninitialized. This return value will overwrite the number of tokens to deposit for the first asset with a strictly smaller amount.

```
function doHardWork(StrategyDhwParameterBag calldata dhwParams) external returns
(DhwInfo memory dhwInfo) {
  _checkRole(ROLE_STRATEGY_REGISTRY, msg.sender);

  // assetsToDeposit[0..token.length-1]: amount of asset i to deposit
  // assetsToDeposit[token.length]: is there anything to deposit
```

```
uint256[] memory assetsToDeposit = new uint256[](dhwParams.assetGroup.length + 1);
unchecked {
  for (uint256 i; i < dhwParams.assetGroup.length; ++i) {
    assetsToDeposit[i] = IERC20(dhwParams.assetGroup[i]).balanceOf(address(this));

    if (assetsToDeposit[i] > 0) {
        ++assetsToDeposit[dhwParams.assetGroup.length];
    }
  }
}
[...]
// - deposit assets into the protocol
_depositToProtocol(dhwParams.assetGroup, assetsToDeposit, dhwParams.slippages);
```

Figure 25.2: A snippet of the doHardWork function in spool-v2-core/Strategy.sol#L71–L85

Exploit Scenario

The `doHardWork` function is called for a smart vault that uses the `ConvexAlusdStrategy` strategy; however, the subsequent call to `_addLiquidity` reverts due to the incorrect number of assets that it is trying to deposit. The smart vault is unusable.

Recommendations

Short term, have `_addLiquidity` loop the `amounts` array for `N_COINS` time instead of its length.

Long term, refactor the `Strategy.doHardWork` function so that it does not use an additional element in the `assetsToDeposit` array to keep track of whether deposits need to be made. Instead, use a separate Boolean variable. The current pattern is too error-prone.

26. Reallocation process reverts when a ghost strategy is present

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-26
Target: `libraries/ReallocationLib.sol`	

Description

The reallocation process reverts in multiple places when a ghost strategy is present. As a result, it is impossible to reallocate a smart vault with a ghost strategy.

The first revert would occur in the `mapStrategies` function (figure 26.1). Users calling the `reallocate` function would not know to add the ghost strategy address in the `strategies` array, which holds the strategies that need to be reallocated. This function reverts if it does not find a strategy in the array. Even if the ghost strategy address is in `strategies`, a revert would occur in the areas described below.

```
   function mapStrategies(
        address[] calldata smartVaults,
        address[] calldata strategies,
        mapping(address => address[]) storage _smartVaultStrategies
    ) private view returns (uint256[][] memory) {
        [...]
            // Loop over smart vault's strategies.
            for (uint256 j; j < smartVaultStrategiesLength; ++j) {
                address strategy = smartVaultStrategies[j];
                bool found = false;

                // Try to find the strategy in the provided list of strategies.
                for (uint256 k; k < strategies.length; ++k) {
                    if (strategies[k] == strategy) {
                        // Match found.
                        found = true;
                        strategyMatched[k] = true;
                        // Add entry to the strategy mapping.
                        strategyMapping[i][j] = k;

                        break;
                    }
                }

                if (!found) {
                    // If a smart vault's strategy was not found in the provided
list
                    // of strategies, this means that the provided list is invalid.
```

```
                    revert InvalidStrategies();
                }
            }
        }
```

Figure 26.1: A snippet of the `mapStrategies` function in spool-v2-core/ReallocationLib.sol#L86–L144

During the reallocation process, the `doReallocation` function calls the `beforeRedeemalCheck` and `beforeDepositCheck` functions even on ghost strategies (figure 26.2); however, their implementation is to revert on ghost strategies with an `IsGhostStrategy` error (figure 26.3).

```
   function doReallocation(
        address[] calldata strategies,
        ReallocationParameterBag calldata reallocationParams,
        uint256[][][] memory reallocationTable
    ) private {
                if (totals[0] == 0) {
                    IStrategy(strategies[i]).beforeRedeemalCheck(0,
reallocationParams.withdrawalSlippages[i]);

                    // There is nothing to withdraw from strategy i.
                    continue;
                }

                // Calculate amount of shares to redeem and to distribute.
                uint256 sharesToDistribute = // first store here total amount of
shares that should have been withdrawn
                 IStrategy(strategies[i]).totalSupply() * totals[0] /
IStrategy(strategies[i]).totalUsdValue();

                IStrategy(strategies[i]).beforeRedeemalCheck(
                    sharesToDistribute, reallocationParams.withdrawalSlippages[i]
                );
                [...]
                // Deposit assets into the underlying protocols.
                for (uint256 i; i < strategies.length; ++i) {
                    IStrategy(strategies[i]).beforeDepositCheck(toDeposit[i],
reallocationParams.depositSlippages[i]);
                [...]
```

Figure 26.2: A snippet of the `doReallocation` function in spool-v2-core/ReallocationLib.sol#L285–L469

```
contract GhostStrategy is IERC20Upgradeable, IStrategy {
   [...]
   function beforeDepositCheck(uint256[] memory, uint256[] calldata) external pure {
        revert IsGhostStrategy();
    }
```

```
    function beforeRedeemalCheck(uint256, uint256[] calldata) external pure {
        revert IsGhostStrategy();
    }
```

Figure 26.3: The `beforeDepositCheck` and `beforeRedeemalCheck` functions in spool-v2-core/GhostStrategy.sol#L98–L104

Exploit Scenario

A strategy is removed from a smart vault. Bob, who has the `ROLE_ALLOCATOR` role, calls `reallocate`, but it reverts and the smart vault is impossible to reallocate.

Recommendations

Short term, modify the associated code so that ghost strategies are not passed to the `reallocate` function in the `_smartVaultStrategies` parameter.

Long term, improve the system's unit and integration tests to test for smart vaults with ghost strategies. Such tests are currently missing.

27. Broken test cases that hide security issues

Severity: **Informational**	Difficulty: **Undetermined**
Type: Testing	Finding ID: TOB-SPL-27
Target: `test/RewardManager.t.sol`, `test/integration/RemoveStrategy.t.sol`	

Description

Multiple test cases do not check sufficient conditions to verify the correctness of the code, which could result in the deployment of buggy code in production and the loss of funds.

The `test_extendRewardEmission_ok` test does not check the new reward rate and duration to verify the effect of the call to the `extendRewardEmission` function on the `RewardManager` contract:

```
function test_extendRewardEmission_ok() public {
    deal(address(rewardToken), vaultOwner, rewardAmount * 2, true);
    vm.startPrank(vaultOwner);
    rewardToken.approve(address(rewardManager), rewardAmount * 2);
    rewardManager.addToken(smartVault, rewardToken, rewardDuration, rewardAmount);

    rewardManager.extendRewardEmission(smartVault, rewardToken, 1 ether,
rewardDuration);
    vm.stopPrank();
}
```

Figure 27.1: An insufficient test case for `extendRewardEmission`
`spool-v2-core/RewardManager.t.sol`

The `test_removeReward_ok` test does not check the new reward token count and the deletion of the reward configuration for the smart vault to verify the effect of the call to the `removeReward` function on the `RewardManager` contract:

```
function test_removeReward_ok() public {
    deal(address(rewardToken), vaultOwner, rewardAmount, true);
    vm.startPrank(vaultOwner);
    rewardToken.approve(address(rewardManager), rewardAmount);
    rewardManager.addToken(smartVault, rewardToken, rewardDuration, rewardAmount);

    skip(rewardDuration + 1);

    rewardManager.removeReward(smartVault, rewardToken);
    vm.stopPrank();
}
```

Figure 27.2: An insufficient test case for `removeReward`
`spool-v2-core/RewardManager.t.sol`

There is no test case to check the access controls of the `removeReward` function. Similarly, the `test_forceRemoveReward_ok` test does not check the effects of the forced removal of a reward token. Findings TOB-SPL-28 and TOB-SPL-29 were not detected by tests because of these broken test cases.

The `test_removeStrategy_betweenFlushAndDHW` test does not check the balance of the master wallet. The `test_removeStrategy_betweenFlushAndDhwWithdrawals` test removes the strategy before the "do hard work" execution of the deposit cycle instead of removing it before the "do hard work" execution of the withdrawal cycle, making this test case redundant. Finding TOB-SPL-33 would have been detected if this test had been correctly implemented.

There may be other broken tests that we did not find, as we could not cover all of the test cases.

Exploit Scenario
The Spool team deploys the protocol. After some time, the Spool team makes some changes in the code that introduces a bug that goes unnoticed due to the broken test cases. The team deploys the new changes with confidence in their tests and ends up introducing a security issue in the production deployment of the protocol.

Recommendations
Short term, fix the test cases described above.

Long term, review all of the system's test cases and make sure that they verify the given state change correctly and sufficiently after an interaction with the protocol. Use Necessist to find broken test cases and fix them.

28. Reward emission can be extended for a removed reward token

Severity: **Medium**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-SPL-28
Target: `rewards/RewardManager.sol`	

Description

Smart vault owners can extend the reward emission for a removed token, which may cause tokens to be stuck in the `RewardManager` contract.

The `removeReward` function in the `RewardManager` contract calls the `_removeReward` function, which does not remove the reward configuration:

```
function _removeReward(address smartVault, IERC20 token) private {
    uint256 _rewardTokensCount = rewardTokensCount[smartVault];
    for (uint256 i; i < _rewardTokensCount; ++i) {
        if (rewardTokens[smartVault][i] == token) {
            rewardTokens[smartVault][i] =
rewardTokens[smartVault][_rewardTokensCount - 1];

            delete rewardTokens[smartVault][_rewardTokensCount- 1];
            rewardTokensCount[smartVault]--;
            emit RewardRemoved(smartVault, token);

            break;
        }
    }
}
```

Figure 28.1: The `_removeReward` function in `spool-v2-core/RewardManger.sol`

The `extendRewardEmission` function checks whether the value of `tokenAdded` in the `rewardConfiguration[smartVault][token]` configuration is not zero to make sure that the token was already added to the smart vault:

```
function extendRewardEmission(address smartVault, IERC20 token, uint256 reward,
uint32 rewardsDuration)
    external
    onlyAdminOrVaultAdmin(smartVault, msg.sender)
    exceptUnderlying(smartVault, token)
{
    if (tokenBlacklist[smartVault][token]) revert
RewardTokenBlacklisted(address(token));
    if (rewardsDuration == 0) revert InvalidRewardDuration();
```

```
    if (rewardConfiguration[smartVault][token].tokenAdded == 0) {
        revert InvalidRewardToken(address(token));
    }

    rewardConfiguration[smartVault][token].rewardsDuration = rewardsDuration;
    _extendRewardEmission(smartVault, token, reward);
}
```

Figure 28.2: The `extendRewardEmission` function in `spool-v2-core/RewardManger.sol`

After removing a reward token from a smart vault, the value of `tokenAdded` in the `rewardConfiguration[smartVault][token]` configuration is left as nonzero, which allows the smart vault owner to extend the reward emission for the removed token.

Exploit Scenario

Alice adds a reward token A to her smart vault S. After a month, she removes token A from her smart vault. After some time, she forgets that she removed token A from her vault. She calls `extendRewardEmission` with 1,000 token A as the reward. The amount of token A is transferred from Alice to the `RewardManager` contract, but it is not distributed to the users because it is not present in the list of reward tokens added for smart vault S. The 1,000 tokens are stuck in the `RewardManager` contract.

Recommendations

Short term, modify the associated code so that it deletes the `rewardConfiguration[smartVault][token]` configuration when removing a reward token for a smart vault.

Long term, add test cases to check for expected user interactions to catch bugs such as this.

29. A reward token cannot be added once it is removed from a smart vault

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-SPL-29
Target: `rewards/RewardManager.sol`	

Description

Smart vault owners cannot add reward tokens again after they have been removed once from the smart vault, making owners incapable of providing incentives to users.

The `removeReward` function in the `RewardManager` contract calls the `_removeReward` function, which does not remove the reward configuration:

```
function _removeReward(address smartVault, IERC20 token) private {
    uint256 _rewardTokensCount = rewardTokensCount[smartVault];
    for (uint256 i; i < _rewardTokensCount; ++i) {
        if (rewardTokens[smartVault][i] == token) {
            rewardTokens[smartVault][i] =
rewardTokens[smartVault][_rewardTokensCount - 1];

            delete rewardTokens[smartVault][_rewardTokensCount- 1];
            rewardTokensCount[smartVault]--;
            emit RewardRemoved(smartVault, token);

            break;
        }
    }
}
```

Figure 29.1: The `_removeReward` function in `spool-v2-core/RewardManger.sol`

The `addToken` function checks whether the value of `tokenAdded` in the `rewardConfiguration[smartVault][token]` configuration is zero to make sure that the token was not already added to the smart vault:

```
function addToken(address smartVault, IERC20 token, uint32 rewardsDuration, uint256
reward)
    external
    onlyAdminOrVaultAdmin(smartVault, msg.sender)
    exceptUnderlying(smartVault, token)
{
    RewardConfiguration storage config = rewardConfiguration[smartVault][token];

    if (tokenBlacklist[smartVault][token]) revert
```

```
RewardTokenBlacklisted(address(token));
    if (config.tokenAdded != 0) revert RewardTokenAlreadyAdded(address(token));
    if (rewardsDuration == 0) revert InvalidRewardDuration();
    if (rewardTokensCount[smartVault] > 5) revert RewardTokenCapReached();

    rewardTokens[smartVault][rewardTokensCount[smartVault]] = token;
    rewardTokensCount[smartVault]++;

    config.rewardsDuration = rewardsDuration;

    if (reward > 0) {
        _extendRewardEmission(smartVault, token, reward);
    }
}
```

Figure 29.2: The addToken function in spool-v2-core/RewardManger.sol

After a reward token is removed from a smart vault, the value of `tokenAdded` in the `rewardConfiguration[smartVault][token]` configuration is left as nonzero, which prevents the smart vault owner from adding the token again for reward distribution as an incentive to the users of the smart vault.

Exploit Scenario

Alice adds a reward token A to her smart vault S. After a month, she removes token A from her smart vault. Noticing the success of her earlier reward incentive program, she wants to add reward token A to her smart vault again, but her transaction to add the reward token reverts, leaving her with no choice but to distribute another token.

Recommendations

Short term, modify the associated code so that it deletes the `rewardConfiguration[smartVault][token]` configuration when removing a reward token for a smart vault.

Long term, add test cases to check for expected user interactions to catch bugs such as this.

30. Missing whenNotPaused modifier

Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-30
Target: `rewards/RewardPool.sol`	

Description
The documentation specifies which functionalities should not be working when the system is paused, including the claiming of rewards; however, the `claim` function does not have the `whenNotPaused` modifier. As a result, users can claim their rewards even when the system is paused.

```
If the system is paused:
   -  users can't claim vault incentives
   -  [...]
```

Figure 30.1: A snippet of the provided Spool documentation

```
   function claim(ClaimRequest[] calldata data) public {
```

Figure 30.2: The `claim` function header in `spool-v2-core/RewardPool.sol#L47`

Exploit Scenario
Alice, who has the `ROLE_PAUSER` role in the system, pauses the protocol after she sees a possible vulnerability in the `claim` function. The Spool team believes there are no possible funds moving from the system; however, users can still claim their rewards.

Recommendations
Short term, add the `whenNotPaused` modifier to the `claim` function.

Long term, improve the system's unit and integration tests by adding a test to verify that the expected functionalities do not work when the system is in a paused state.

31. Users who deposit and then withdraw before doHardWork lose their tokens

Severity: **High**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-SPL-31
Target: `managers/DepositManager.sol`	

Description

Users who deposit and then withdraw assets before `doHardWork` is called will receive zero tokens from their withdrawal operations.

When a user deposits assets, the `depositAssets` function mints an NFT with some metadata to the user who can later redeem it for the underlying SVT tokens.

```
   function depositAssets(DepositBag calldata bag, DepositExtras calldata bag2)
        external
        onlyRole(ROLE_SMART_VAULT_MANAGER, msg.sender)
        returns (uint256[] memory, uint256)
    {
      [...]
      // mint deposit NFT
      DepositMetadata memory metadata = DepositMetadata(bag.assets, block.timestamp,
bag2.flushIndex);
      uint256 depositId = ISmartVault(bag.smartVault).mintDepositNFT(bag.receiver,
metadata);
      [...]
    }
```

Figure 31.1: A snippet of the `depositAssets` function in `spool-v2-core/DepositManager.sol#L379–L439`

Users call the `claimSmartVaultTokens` function in the `SmartVaultManager` contract to claim SVT tokens. It is important to note that this function calls the `_syncSmartVault` function with `false` as the last argument, which means that it will not revert if the current flush index and the flush index to sync are the same. Then, `claimSmartVaultTokens` delegates the work to the corresponding function in the `DepositManager` contract.

```
   function claimSmartVaultTokens(address smartVault, uint256[] calldata nftIds,
uint256[] calldata nftAmounts)
        public
        whenNotPaused
        returns (uint256)
    {
```

```
        _onlyRegisteredSmartVault(smartVault);
        address[] memory tokens =
_assetGroupRegistry.listAssetGroup(_smartVaultAssetGroups[smartVault]);
        _syncSmartVault(smartVault, _smartVaultStrategies[smartVault], tokens,
false);
        return _depositManager.claimSmartVaultTokens(smartVault, nftIds, nftAmounts,
tokens, msg.sender);
    }
```

Figure 31.2: A snippet of the `claimSmartVaultTokens` function in spool-v2-core/SmartVaultManager.sol#L238–L247

Later, the `claimSmartVaultTokens` function in `DepositManager` (figure 31.3) computes the SVT tokens that users will receive by calling the `getClaimedVaultTokensPreview` function and passing the `bag.mintedSVTs` value for the flush corresponding to the burned NFT.

```
   function claimSmartVaultTokens(
        address smartVault,
        uint256[] calldata nftIds,
        uint256[] calldata nftAmounts,
        address[] calldata tokens,
        address executor
    ) external returns (uint256) {
        _checkRole(ROLE_SMART_VAULT_MANAGER, msg.sender);
        [...]
        ClaimTokensLocalBag memory bag;
        ISmartVault vault = ISmartVault(smartVault);
        bag.metadata = vault.burnNFTs(executor, nftIds, nftAmounts);

        for (uint256 i; i < nftIds.length; ++i) {
            if (nftIds[i] > MAXIMAL_DEPOSIT_ID) {
                revert InvalidDepositNftId(nftIds[i]);
            }

            // we can pass empty strategy array and empty DHW index array,
            // because vault should already be synced and mintedVaultShares values
available
            bag.data = abi.decode(bag.metadata[i], (DepositMetadata));
            bag.mintedSVTs =
_flushShares[smartVault][bag.data.flushIndex].mintedVaultShares;

            claimedVaultTokens +=
                getClaimedVaultTokensPreview(smartVault, bag.data, nftAmounts[i],
bag.mintedSVTs, tokens);
        }
```

Figure 31.3: A snippet of the `claimSmartVaultTokens` in spool-v2-core/DepositManager.sol#L135–L184

Then, `getClaimedVaultTokensPreview` calculates the SVT tokens proportional to the amount deposited.

```
   function getClaimedVaultTokensPreview(
        address smartVaultAddress,
        DepositMetadata memory data,
        uint256 nftShares,
        uint256 mintedSVTs,
        address[] calldata tokens
    ) public view returns (uint256) {
        [...]

        for (uint256 i; i < data.assets.length; ++i) {
            depositedUsd += _priceFeedManager.assetToUsdCustomPrice(tokens[i],
data.assets[i], exchangeRates[i]);
            totalDepositedUsd +=
                _priceFeedManager.assetToUsdCustomPrice(tokens[i],
totalDepositedAssets[i], exchangeRates[i]);
        }
        uint256 claimedVaultTokens = mintedSVTs * depositedUsd / totalDepositedUsd;

        return claimedVaultTokens * nftShares / NFT_MINTED_SHARES;
    }
```

Figure 31.4: A snippet of the `getClaimedVaultTokensPreview` function in spool-v2-core/DepositManager.sol#L546–L572

However, the value of `_flushShares[smartVault][bag.data.flushIndex].mintedVaultShares`, shown in figure 31.3, will always be `0`: the value is updated in the `syncDeposit` function, but because the current flush cycle is not finished yet, `syncDeposit` cannot be called through `syncSmartVault`.

The same problem appears in the `redeem`, `redeemFast`, and `claimWithdrawal` functions.

Exploit Scenario

Bob deposits assets into a smart vault, but he notices that he deposited in the wrong smart vault. He calls `redeem` and `claimWithdrawal`, expecting to receive back his tokens, but he receives zero tokens. The tokens are locked in the smart contracts.

Recommendations

Short term, do not allow users to withdraw tokens when the corresponding flush has not yet happened.

Long term, document and test the expected effects when calling functions in all of the possible orders, and add adequate constraints to avoid unexpected behavior.

32. Lack of events emitted for state-changing functions

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-SPL-32
Target: `src/`	

Description

Multiple critical operations do not emit events. As a result, it will be difficult to review the correct behavior of the contracts once they have been deployed.

Events generated during contract execution aid in monitoring, baselining of behavior, and detection of suspicious activity. Without events, users and blockchain-monitoring systems cannot easily detect behavior that falls outside the baseline conditions. This may prevent malfunctioning contracts or attacks from being detected.

The following operations should trigger events:

- `SpoolAccessControl.grantSmartVaultOwnership`
- `ActionManager.setActions`
- `SmartVaultManager.registerSmartVault`
- `SmartVaultManager.removeStrategy`
- `SmartVaultManager.syncSmartVault`
- `SmartVaultManager.reallocate`
- `StrategyRegistry.registerStrategy`
- `StrategyRegistry.removeStrategy`
- `StrategyRegistry.doHardWork`
- `StrategyRegistry.setEcosystemFee`
- `StrategyRegistry.setEcosystemFeeReceiver`
- `StrategyRegistry.setTreasuryFee`
- `StrategyRegistry.setTreasuryFeeReceiver`

- `Strategy.doHardWork`
- `RewardManager.addToken`
- `RewardManager.extendRewardEmission`

Exploit Scenario
The Spool system experiences a security incident, but the Spool team has trouble reconstructing the sequence of events causing the incident because of missing log information.

Recommendations
Short term, add events for all operations that may contribute to a higher level of monitoring and alerting.

Long term, consider using a blockchain-monitoring system to track any suspicious behavior in the contracts. The system relies on several contracts to behave as expected. A monitoring mechanism for critical events would quickly detect any compromised system components.

33. Removal of a strategy could result in loss of funds

Severity: **Medium**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-SPL-33
Target: `managers/SmartVaultManager.sol`	

Description
A Spool admin can remove a strategy from the system, which will be replaced by a ghost strategy in all smart vaults that use it; however, if a strategy is removed when the system is in specific states, funds to be deposited or withdrawn in the next “do hard work” cycle will be lost.

If the following sequence of events occurs, the asset deposited will be lost from the removed strategy:

1. A user deposits assets into a smart vault.
2. The flush function is called. The `StrategyRegistry._assetsDeposited[strategy][xxx][yyy]` storage variable now has assets to send to the given strategy in the next “do hard work” cycle.
3. The strategy is removed.
4. `doHardWork` is called, but the assets for the removed strategy are locked in the master wallet because the function can be called only for valid strategies.

If the following sequence of events occurs, the assets withdrawn from a removed strategy will be lost:

1. `doHardWork` is called.
2. The strategy is removed before a smart vault sync is done.

Exploit Scenario
Multiple smart vaults use strategy A. Users deposited a total of $1 million, and $300,000 should go to strategy A. Strategy A is removed due to an issue in the third-party protocol. All of the $300,000 is locked in the master wallet.

Recommendations

Short term, modify the associated code to properly handle deposited and withdrawn funds when strategies are removed.

Long term, improve the system's unit and integration tests: consider all of the possible transaction sequences in the system's state and test them to ensure their correct behavior.

34. ExponentialAllocationProvider reverts on strategies without risk scores

Severity: **Medium**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-SPL-34
Target: `providers/ExponentialAllocationProvider.sol`	

Description
The `ExponentialAllocationProvider.calculateAllocation` function can revert due to division-by-zero error when a strategy's risk score has not been set by the risk provider.

The `risk` variable in `calculateAllocation` represents the risk score set by the risk provider for the given strategy, represented by the index `i`. Ghost strategies can be passed to the function. If a ghost strategy's risk score has not been set (which is likely, as there would be no reason to set one), the function will revert with a division-by-zero error.

```
   function calculateAllocation(AllocationCalculationInput calldata data) external
pure returns (uint256[] memory) {
        if (data.apys.length != data.riskScores.length) {
            revert ApysOrRiskScoresLengthMismatch(data.apys.length,
data.riskScores.length);
        }
        [...]
        for (uint8 i; i < data.apys.length; ++i) {
            [...]
            int256 risk = fromUint(data.riskScores[i]);

            results[i] = uint256(div(apy, risk));

            resultSum += results[i];
        }
```

Figure 34.1: A snippet of the `calculateAllocation` function in spool-v2-core/ExponentialAllocationProvider.sol#L309–L340

Exploit Scenario
A strategy is removed from a smart vault that uses the `ExponentialAllocationProvider` contract. Bob, who has the `ROLE_ALLOCATOR` role, calls `reallocate`; however, it reverts, and the smart vault is impossible to reallocate.

Recommendations
Short term, modify the `calculateAllocation` function so that it properly handles strategies with uninitialized risk scores.

Long term, improve the unit and integration tests for the allocators. Refactor the codebase so that ghost strategies are not passed to the `calculateAllocator` function.

35. Removing a strategy makes the smart vault unusable

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-35
Target: `managers/DepositManager.sol`	

Description

Removing a strategy from a smart vault causes every subsequent deposit transaction to revert, making the smart vault unusable.

The `deposit` function of the `SmartVaultManager` contract calls the `depositAssets` function on the `DepositManager` contract. The `depositAssets` function calls the `checkDepositRatio` function, which takes an argument called `strategyRatios`:

```
function depositAssets(DepositBag calldata bag, DepositExtras calldata bag2)
    external
    onlyRole(ROLE_SMART_VAULT_MANAGER, msg.sender)
    returns (uint256[] memory, uint256)
{
    ...

    // check if assets are in correct ratio
    checkDepositRatio(
        bag.assets,
        SpoolUtils.getExchangeRates(bag2.tokens, _priceFeedManager),
        bag2.allocations,
        SpoolUtils.getStrategyRatiosAtLastDhw(bag2.strategies, _strategyRegistry)
    );

    ...

    return
(_vaultDeposits[bag.smartVault][bag2.flushIndex].toArray(bag2.tokens.length),
depositId);
}
```

Figure 35.1: The `depositAssets` function in `spool-v2-core/DepositManager.sol`

The value of `strategyRatios` is fetched from the `StrategyRegistry` contract, which returns an empty array on ghost strategies. This empty array is then used in a `for` loop in the `calculateFlushFactors` function:

```
function calculateFlushFactors(
    uint256[] memory exchangeRates,
```

```
    uint16a16 allocation,
    uint256[][] memory strategyRatios
) public pure returns (uint256[][] memory) {
    uint256[][] memory flushFactors = new uint256[][](strategyRatios.length);

    // loop over strategies
    for (uint256 i; i < strategyRatios.length; ++i) {
        flushFactors[i] = new uint256[](exchangeRates.length);

        uint256 normalization = 0;
        // loop over assets
        for (uint256 j = 0; j < exchangeRates.length; j++) {
            normalization += strategyRatios[i][j] * exchangeRates[j];
        }

        // loop over assets
        for (uint256 j = 0; j < exchangeRates.length; j++) {
            flushFactors[i][j] = allocation.get(i) * strategyRatios[i][j] *
PRECISION_MULTIPLIER / normalization;
        }
    }

    return flushFactors;
}
```

Figure 35.2: The `calculateFlushFactors` function in `spool-v2-core/DepositManager.sol`

The statement calculating the value of `normalization` tries to access an index of the empty array and reverts with the `Index out of bounds` error, causing the `deposit` function to revert for every transaction thereafter.

Exploit Scenario

A Spool admin removes a strategy from a smart vault. Because of the presence of a ghost strategy, users' deposit transactions into the smart vault revert with the `Index out of bounds` error.

Recommendations

Short term, modify the `calculateFlushFactors` function so that it skips ghost strategies in the loop used to calculate the value of `normalization`.

Long term, review the entire codebase, check the effects of removing strategies from smart vaults, and ensure that all of the functionality works for smart vaults with one or more ghost strategies.

36. Issues with the management of access control roles in deployment script

Severity: **Low**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-SPL-36
Target: `script/DeploySpool.s.sol`	

Description
The deployment script does not properly manage or assign access control roles. As a result, the protocol will not work as expected, and the protocol's contracts cannot be upgraded.

The deployment script has multiple issues regarding the assignment or transfer of access control roles. It fails to grant certain roles and to revoke temporary roles on deployment:

- Ownership of the `ProxyAdmin` contract is not transferred to an EOA, multisig wallet, or DAO after the system is deployed, making the smart contracts non-upgradeable.
- The `DEFAULT_ADMIN_ROLE` role is not transferred to an EOA, multisig wallet, or DAO after the system is deployed, leaving no way to manage roles after deployment.
- The `ADMIN_ROLE_STRATEGY` role is not assigned to the `StrategyRegistry` contract, which is required to grant the `ROLE_STRATEGY` role to a strategy contract. Because of this, new strategies cannot be registered.
- The `ADMIN_ROLE_SMART_VAULT_ALLOW_REDEEM` role is not assigned to the `SmartVaultFactory` contract, which is required to grant the `ROLE_SMART_VAULT_ALLOW_REDEEM` role to `smartVault` contracts.
- The `ROLE_SMART_VAULT_MANAGER` and `ROLE_MASTER_WALLET_MANAGER` roles are not assigned to the `DepositManager` and `WithdrawalManager` contracts, making them unable to move funds from the master wallet contract.

We also found that the `ROLE_SMART_VAULT_ADMIN` role is not assigned to the smart vault owner when a new smart vault is created. This means that smart vault owners will not be able to manage their smart vaults.

Exploit Scenario
The Spool team deploys the smart contracts using the deployment script, but due to the issues described in this finding, the team is not able to perform the role management and upgrades when required.

Recommendations

Short term, modify the deployment script so that it does the following on deployment:

- Transfers ownership of the `proxyAdmin` contract to an EOA, multisig wallet, or DAO
- Transfers the `DEFAULT_ADMIN_ROLE` role to an EOA, multisig wallet, or DAO
- Grants the required roles to the smart contracts
- Allow the `SmartVaultFactory` contract to grant the `ROLE_SMART_VAULT_ADMIN` role to owners of newly created smart vaults

Long term, document all of the system's roles and interactions between components that require privileged roles. Make sure that all of the components are granted their required roles following the principle of least privilege to keep the protocol secure and functioning as expected.

37. Risk of DoS due to unbounded loops

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-SPL-37
Target: `managers/SmartVaultManager.sol`	

Description

Guards and actions are run in unbounded loops. A smart vault creator can add too many guards and actions, potentially trapping the deposit and withdrawal functionality due to a lack of gas.

The `runGuards` function calls all the configured guard contracts in a loop:

```
function runGuards(address smartVaultId, RequestContext calldata context) external
view {
    if (guardPointer[smartVaultId][context.requestType] == address(0)) {
        return;
    }

    GuardDefinition[] memory guards = _readGuards(smartVaultId,
context.requestType);

    for (uint256 i; i < guards.length; ++i) {
        GuardDefinition memory guard = guards[i];

        bytes memory encoded = _encodeFunctionCall(smartVaultId, guard, context);
        (bool success, bytes memory data) =
guard.contractAddress.staticcall(encoded);
        _checkResult(success, data, guard.operator, guard.expectedValue, i);
    }
}
```

Figure 37.1: The `runGuards` function in `spool-v2-core/GuardManager.sol`

Multiple conditions can cause this loop to run out of gas:

- The vault creator adds too many guards.
- One of the guard contracts consumes a high amount of gas.
- A guard starts consuming a high amount of gas after a specific block or at a specific state.

If user transactions reach out-of-gas errors due to these conditions, smart vaults can become unusable, and funds can become stuck in the protocol.

A similar issue affects the `runActions` function in the `AuctionManager` contract.

Exploit Scenario

Eve creates a smart vault with an upgradeable guard contract. Later, when users have made large deposits, Eve upgrades the guard contract to consume all of the available gas to trap user deposits in the smart vault for as long as she wants.

Recommendations

Short term, model all of the system's variable-length loops, including the ones used by `runGuards` and `runActions`, to ensure they cannot block contract execution within expected system parameters.

Long term, carefully audit operations that consume a large amount of gas, especially those in loops.

38. Unsafe casts throughout the codebase

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-SPL-38
Target: `src/`	

Description

The codebase contains unsafe casts that could cause mathematical errors if they are reachable in certain states.

Examples of possible unsafe casts are shown in figures 38.1 and 38.2.

```
    function flushSmartVault(
        address smartVault,
        uint256 flushIndex,
        address[] calldata strategies,
        uint16a16 allocation,
        address[] calldata tokens
    ) external returns (uint16a16) {
        [...]
        _flushShares[smartVault][flushIndex].flushSvtSupply =
uint128(ISmartVault(smartVault).totalSupply());

        return _strategyRegistry.addDeposits(strategies, distribution);
    }
```

Figure 38.1: A possible unsafe cast in `spool-v2-core/DepositManager.sol#L220`

```
    function syncDeposits(
        address smartVault,
        uint256[3] calldata bag,
        // uint256 flushIndex,
        // uint256 lastDhwSyncedTimestamp,
        // uint256 oldTotalSVTs,
        address[] calldata strategies,
        uint16a16[2] calldata dhwIndexes,
        address[] calldata assetGroup,
        SmartVaultFees calldata fees
    ) external returns (DepositSyncResult memory) {
        [...]
        if (syncResult.mintedSVTs > 0) {
            _flushShares[smartVault][bag[0]].mintedVaultShares =
uint128(syncResult.mintedSVTs);
            [...]
        }
```

```
        return syncResult;
    }
```

Figure 38.2: A possible unsafe cast in `spool-v2-core/DepositManager.sol#L243`

Recommendations

Short term, review the codebase to identify all of the casts that may be unsafe. Analyze whether these casts could be a problem in the current codebase and, if they are unsafe, make the necessary changes to make them safe.

Long term, when implementing potentially unsafe casts, always include comments to explain why those casts are safe in the context of the codebase.

Summary of Recommendations

The Spool DAO's Spool V2 project is a work in progress with multiple planned iterations. Trail of Bits recommends that the Spool DAO address the findings detailed in this report and take the following additional steps prior to deployment:

- Improve the system's unit and integration tests before developing more strategies and features.
- Define the expected state transitions of the protocol's different phases and the order in which the protocol's phases can be called. Ensure that the implementation follows the specification.
- Deploy the protocol on the testnet and test all of the admin functions with the entire lifecycle of the protocol before deployment.
- Have another security review conducted on the protocol before deployment.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Transaction Ordering Risks	The system's resilience against malicious ordering of the transactions.
Low-Level Manipulation	The use of low-level operations, including but not limited to assembly code, bitwise operations, and manual calldata construction.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.

Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Findings

This appendix lists findings that are not associated with specific vulnerabilities.

- **Incorrect comments**
  - `StrategyRegistry:L61–L65`: The comment indicates that the `_dhwAssetRatios` variable stores the exchange rate, but it does not.
  - `Strategy.sol:L98–L99`: The comment indicates that the `_collectPlatformFees` function mints SVTs, but it actually mints SSTs.
  - `Strategy.sol:L332–L335`: The comment indicates that the recipient of assets transferred from the `_redeemShares` function is the master wallet, but this is not always the case.
  - `DepositManager.sol:L574–L595`: The `@return` comment regarding array dimensions on line 577 is incorrect.
  - `CompoundV2Strategy.sol:L23–L24`: The comment indicates that the `swapper` variable stores the comptroller address, but it stores the swapper address.
- **Unclear events**
  - `SpoolAccessControl.sol:L50–L62`: The `RoleGranted` and `RoleRevoked` events are emitted when roles are granted or revoked, but they do not indicate which vault was modified and what role was granted or revoked.
- **Unreachable or unnecessary code**
  - `ActionManager.sol:L4`: The imported `@solmate/utils/SSTORE2.sol` file is not used in the code.
  - `ActionManager.sol:L17–L18`: The `actionsExist` mapping is private, so it is not accessed from other contracts. Additionally, its values are set in only one place in the contract. This mapping can be replaced with a private function:

```
function actionsExist(address smartVault, RequestType requestType)
private view returns (bool) {
    return actions[smartVault][requestType].length > 0
}
```

Figure C.1: A proposed private function to replace the `actionsExist` mapping

- SmartVault.sol:L32–L36: The `_activeUserNFTIds` and `_activeUserNFTCount` mappings could be replaced with a single mapping:

```
mapping(address => uint256[]) private _activeUserNFTIds;
```

Figure C.2: A proposed single mapping to replace `_activeUserNFTIds` and `_activeUserNFTCount`

- DepositSwap.sol:L95–L98: The `Swapper` contract will never return ETH, so this code is unreachable.

- **Incorrect, inconsistent, and missing checks**
  - In most functions, access controls are enforced via the `onlyRole()` modifier. However, there are some cases in the following files in which `_checkRole()` is used instead: SpoolAccessControl.sol, SpoolAccessControllable.sol, DepositManager.sol, SmartVaultManager.sol, StrategyRegistry.sol, and Strategy.sol. Access control methods should be consistent throughout the codebase.
  - SmartVault.sol:L136: The check in this location should use > instead of >=. However, this incorrect check is not a security issue, as the value of `MAX_DEPOSIT_ID` is highly unlikely to have such a high number of deposits.
  - SmartVaultManager.sol:L238–L247: The `claimSmartVaultTokens` function is missing a check to ensure that the `nftIds` and `nftAmounts` arrays are the same length.
  - SmartVaultManager.sol:L623–L641: The `_redeem` function is missing a check to ensure that the `bag.nftIds` and `bag.nftAmounts` arrays are the same length.
  - deposit.t.sol:L114: The check should be against the `StrategyC` balance of `StrategyC`, rather than the current check against `StrategyB`.

- **General coding recommendations**
  - SpoolAccessControl.sol:L19–L24: The `AccessControlUpgradeable` contract is not initialized because the `__AccessControl_init` function is not called. This is not a security issue in this particular codebase because the initializer is empty, but it is a good practice to call all initializers.
  - SpoolAccessControl.sol:L19–L24: The `_setupRole()` function is deprecated by OpenZeppelin. OpenZeppelin's current recommendation for

setting roles is to use the `_grantRole()` function, as stated in the documentation for `AccessControlUpgradeable`.

- `LinearAllocationProvider.sol:L8`: To maintain consistency with the filename and other allocation provider contracts, the contract should be renamed to `LinearAllocationProvider`.
- `SmartVault.sol:L146–L162`: For clarity, the `mint()` and `burn()` functions should have more descriptive names, such as `mintVaultShares()` and `burnVaultShares()`.
- `Strategy.sol:L74–L87`: The use of a new element at the end of the `assetsToDeposit` array as a signaling mechanism can introduce bugs, since the array is passed as a parameter to other functions, some of which work with two arrays that should be of the same length. Instead, a Boolean variable should be used to indicate whether there are assets to be deposited. A similar issue appears in `ReallocationLib.sol:L448–L450`.
- `DepositSwap.sol:L88–L93`: The contract receives ETH and returns WETH. The returned amount should be in the same asset that was received.
- `ReallocationLib.sol:L356`: This line could be rewritten to reuse the `totalUnmatchedWithdrawals` variable that is calculated earlier.

D. Automated Analysis Tool Configuration

Slither

We used Slither to detect common issues and anti-patterns in the codebase. Slither discovered a number of low- and medium-severity issues, such as TOB-SPL-4 and TOB-SPL-3. Integrating Slither into the project's testing environment can help find similar issues and improve the overall quality of the smart contracts' code.

```
slither --compile-force-framework foundry --filter-paths test .
```

Figure D.1: An example Slither configuration

We also found that the `filter-paths` parameter of the `slither.config.json` file is set to "`lib`", which prevents Slither from reporting issues in the smart contracts in the `src/libraries` directory. This is because Slither uses a regex pattern to match the source path of a file with the `filter-paths` parameter.

Necessist

We used Necessist to identify insufficient, incorrect, and broken test cases. Broken test cases provide a false sense of confidence in the code's correctness; therefore, it is important to identify and fix these tests to prevent security incidents.

```
necessist
```

Figure D.2: An example command to run Necessist

Necessist found around 300 occurrences in which removing a line from a test case did not fail the test case. We analyzed a few of these occurrences and found some informational- and low-severity issues, such as TOB-SPL-27, TOB-SPL-28, and TOB-SPL-29. Integrating Necessist into the project's testing environment can help find similar issues and improve the overall quality of the test cases.

E. Proofs of Concept

This appendix includes proofs of concept to demonstrate findings TOB-SPL-31 and TOB-SPL-33.

Proof of Concept for TOB-SPL-31

The `test_burnDNFTwithZeroClaim` function demonstrates that it is possible for a user to burn a D-NFT without receiving any SVTs. This test function can be added to the `test/integration/deposit.t.sol` file to run it with the test suit:

```
function test_burnDNFTwithZeroClaim() public {
    // Alice deposits
    vm.startPrank(alice);

    uint256[] memory depositAmounts = Arrays.toArray(100 ether, 7.237 ether, 438.8
ether);

    tokenA.approve(address(smartVaultManager), depositAmounts[0]);
    tokenB.approve(address(smartVaultManager), depositAmounts[1]);
    tokenC.approve(address(smartVaultManager), depositAmounts[2]);

    uint256 aliceDepositNftId =
smartVaultManager.deposit(DepositBag(address(smartVault), depositAmounts, alice,
address(0), false));

    vm.stopPrank();

    // flush
    smartVaultManager.flushSmartVault(address(smartVault));
    assertEq(smartVault.balanceOfFractional(alice, aliceDepositNftId),
NFT_MINTED_SHARES);

    uint256[] memory amounts = Arrays.toArray(NFT_MINTED_SHARES);
    uint256[] memory ids = Arrays.toArray(aliceDepositNftId);
    vm.prank(alice);
    smartVaultManager.claimSmartVaultTokens(address(smartVault), ids, amounts);

    // check state
    // - 0 vault tokens were claimed successfully
    assertEq(smartVault.balanceOf(address(alice)), 0);
    assertEq(smartVault.balanceOf(address(smartVault)), 0);
    // - deposit NFT was burned
    assertEq(smartVault.balanceOfFractional(alice, aliceDepositNftId), 0);
}
```

Figure E.1: A proof of concept for TOB-SPL-31

Similar test cases can be written to demonstrate that burning a W-NFT may result in a zero-value withdrawal and that the `redeemFast` function may result in a zero-value SVT claim and withdrawal.

Proof of Concept for TOB-SPL-33

The `test_removeStrategy_betweenFlushAndDHW` function demonstrates that deposited funds could become stuck in the `MasterWallet` contract after a strategy is removed. This test can be added to the `test/integration/RemoveStrategy.t.sol` file to run it with the test suite:

```
function test_removeStrategy_betweenFlushAndDHW() public {
    TestBag memory bag;
    createVault();
    vm.clearMockedCalls();

    bag.fees = SmartVaultFees(0, 0, 0);
    bag.dhwSwapInfo = new SwapInfo[][](3);
    bag.depositAmounts = Arrays.toArray(100 ether, 7.237 ether, 438.8 ether);

    vm.prank(alice);
    smartVaultManager.deposit(DepositBag(address(smartVault), bag.depositAmounts,
alice, address(0), true));

    smartVaultManager.removeStrategy(smartVaultStrategies[0], true);
    smartVaultStrategies = smartVaultManager.strategies(address(smartVault));

    vm.startPrank(doHardWorker);
    strategyRegistry.doHardWork(
        generateDhwParameterBag(Arrays.toArray(address(strategyB),
address(strategyC)), assetGroup)
    );
    vm.stopPrank();

    DepositSyncResult memory syncResult = depositManager.syncDepositsSimulate(
        SimulateDepositParams(
            address(smartVault),
            [uint256(0), 0, 0], // flush index, first dhw timestamp, total SVTs
minted till now
            smartVaultStrategies,
            assetGroup,
            Arrays.toUint16a16(1, 1, 1),
            Arrays.toUint16a16(0, 0, 0),
            bag.fees
        )
    );
    smartVaultManager.syncSmartVault(address(smartVault), true);

    assertEq(smartVaultStrategies[0], address(ghostStrategy));
    assertEq(syncResult.mintedSVTs, smartVault.totalSupply());
    assertEq(ghostStrategy.totalSupply(), 0);
```

```
    assertEq(strategyA.totalSupply(), syncResult.sstShares[0]);
    assertEq(strategyB.totalSupply(), syncResult.sstShares[1]);
    assertEq(strategyC.totalSupply(), syncResult.sstShares[2]);
    assertEq(syncResult.sstShares[0], 0);
    assertEq(syncResult.sstShares[1], 10714883153826625699325000);
    assertEq(syncResult.sstShares[2], 35716275414794966628800000);
    // Check funds stuck in the MasterWallet
    assertGt(tokenA.balanceOf(address(masterWallet)), 0);
    assertGt(tokenB.balanceOf(address(masterWallet)), 0);
    assertGt(tokenC.balanceOf(address(masterWallet)), 0);
}
```

Figure E.2: A proof of concept for TOB-SPL-33

A similar test case can be written to demonstrate that funds withdrawn by users could become stuck in the `MasterWallet` contract.

F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From May 1 to May 2, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Spool team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 38 issues described in this report, Spool has resolved 34, has partially resolved three, and has not resolved the remaining issue. For additional information, refer to the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Solidity compiler optimizations can be problematic	Undetermined	Unresolved
2	Risk of SmartVaultFactory DoS due to lack of access controls on grantSmartVaultOwnership	High	Resolved
3	Lack of zero-value check on constructors and initializers	Medium	Resolved
4	Upgradeable contracts set state variables in the constructor	Medium	Resolved
5	Insufficient validation of oracle price data	Low	Resolved
6	Incorrect handling of fromVaultsOnly in removeStrategy	Low	Resolved
7	Risk of LinearAllocationProvider and ExponentialAllocationProvider reverts due to division by zero	Medium	Resolved
8	Strategy APYs are never updated	Medium	Resolved

9	Incorrect bookkeeping of assets deposited into smart vaults	High	Resolved
10	Risk of malformed calldata of calls to guard contracts	Low	Partially resolved
11	GuardManager does not account for all possible types when encoding guard arguments	Low	Partially resolved
12	Use of encoded values in guard contract comparisons could lead to opposite results	Low	Resolved
13	Lack of contract existence checks on low-level calls	Low	Resolved
14	Incorrect use of exchangeRates in doHardWork	High	Resolved
15	LinearAllocationProvider could return an incorrect result	Medium	Resolved
16	Incorrect formula used for adding/subtracting two yields	Medium	Resolved
17	Smart vaults with re-registered strategies will not be usable	Low	Resolved
18	Incorrect handling of partially burned NFTs results in incorrect SVT balance calculation	Low	Resolved
19	Transfers of D-NFTs result in double counting of SVT balance	Medium	Resolved
20	Flawed loop for syncing flushes results in higher management fees	Medium	Resolved
21	Incorrect ghost strategy check	Informational	Resolved
22	Reward configuration not initialized properly when reward is zero	Low	Partially Resolved

23	Missing function for removing reward tokens from the blacklist	Informational	Resolved
24	Risk of unclaimed shares due to loss of precision in reallocation operations	Informational	Resolved
25	Curve3CoinPoolAdapter's _addLiquidity reverts due to incorrect amounts deposited	Medium	Resolved
26	Reallocation process reverts when a ghost strategy is present	High	Resolved
27	Broken test cases that hide security issues	Informational	Resolved
28	Reward emission can be extended for a removed reward token	Medium	Resolved
29	A reward token cannot be added once it is removed from a smart vault	Low	Resolved
30	Missing whenNotPaused modifier	Low	Resolved
31	Users who deposit and then withdraw before doHardWork lose their tokens	High	Resolved
32	Lack of events emitted for state-changing functions	Informational	Resolved
33	Removal of a strategy could result in loss of funds	Medium	Resolved
34	ExponentialAllocationProvider reverts on strategies without risk scores	Medium	Resolved
35	Removing a strategy makes the smart vault unusable	Medium	Resolved
36	Issues with the management of access control roles in deployment script	Low	Resolved

37	Risk of DoS due to unbounded loops	Medium	Resolved
38	Unsafe casts throughout the codebase	Undetermined	Resolved

Detailed Fix Review Results

TOB-SPL-1: Solidity compiler optimizations can be problematic
Unresolved. The client provided the following context for this finding's fix status:

> *Although we acknowledge that compiler optimization could be problematic, due to latent bugs in the optimizer, compiler optimization has become the industry standard.*
>
> *Furthermore, removing compiler optimization at this point would have a considerable impact on the codebase. Some smart contracts would exceed the size limitations and would require extensive refactoring.*

TOB-SPL-2: Risk of SmartVaultFactory DoS due to lack of access controls on grantSmartVaultOwnership
Resolved in PR #158. The `onlyRole` modifier was added to the `grantSmartVaultOwnership` function in the `SpoolAccessControl` contract, as recommended. The `ROLE_SMART_VAULT_INTEGRATOR` role is now needed to grant ownership of a smart vault. Additionally, a new test was added to cover the case in which a non-privileged user calls the function, and changes were made to the test file to ensure that roles are assigned.

TOB-SPL-3: Lack of zero-value check on constructors and initializers
Resolved in PR #159. Zero-address checks were implemented for the constructors and initializers indicated in the finding description and for one state-changing function (`_allowToken` in `AssetGroupRegistry`). All checks in the modified functions revert on zero values with a new `ConfigurationAddressZero` custom error.

TOB-SPL-4: Upgradeable contracts set state variables in the constructor
Resolved in PR #162. All recommended changes were implemented by the Spool team. Additionally, all relevant tests were modified to reflect the new changes.

TOB-SPL-5: Insufficient validation of oracle price data
Resolved in PR #163. The `latestRoundData` function's `roundId`, `updatedAt`, and `answeredInRound` return values are now validated, ensuring that the prices received from the oracle are current.

TOB-SPL-6: Incorrect handling of fromVaultsOnly in removeStrategy
Resolved in PR #164 and PR #195. The `fromVaultsOnly` parameter was renamed to `disableStrategy`, and the `removeStrategy` function was renamed to `removeStrategyFromVaults`. The incorrect check described in the finding was corrected, and the code is now more readable thanks to the name changes. All of the integration tests that call the `removeStrategy` function were refactored with the new function name and parameters.

TOB-SPL-7: Risk of LinearAllocationProvider and ExponentialAllocationProvider reverts due to division by zero
Resolved in PR #214 and PR #220. Checks were added to the `calculateAllocation` functions of the `AllocationProviderLinear` and `ExponentialAllocationProvider` contracts that will skip the division operation if the APY is zero. Additionally, the code handling APY numerical precision was corrected, minor gas optimizations were added, and a final check was added that sets allocations to be equally distributed if all final allocations are zero. The relevant tests in the `AllocationProvider` test suite were modified accordingly.

TOB-SPL-8: Strategy APYs are never updated
Resolved in PR #166. The missing call to the `_updateDhwYieldAndApy` function was added to the `doHardWork` function in the `StrategyRegistry` contract.

TOB-SPL-9: Incorrect bookkeeping of assets deposited into smart vaults
Resolved in PR #165. The `depositAssets` function in the `DepositManager` contract was modified to correctly accumulate and keep track of the assets deposited by users. The deposits array return value and all uses of it in the `SmartVaultManager` contract were removed. A new test case was added to the deposit integration tests to ensure that the system can handle multiple deposits.

TOB-SPL-10: Risk of malformed calldata of calls to guard contracts
Partially resolved in PR #193. The NatSpec documentation for the `GuardDefinition` structure and `IGuardManager` interface was updated to specify the limitations of the values that must be passed to the `methodParamValues` array. No further changes were made to the contract.

TOB-SPL-11: GuardManager does not account for all possible types when encoding guard arguments
Partially resolved in PR #193. The same NatSpec documentation updates that were implemented to mitigate finding TOB-SPL-10 also mitigate this finding.

TOB-SPL-12: Use of encoded values in guard contract comparisons could lead to opposite results
Resolved in PR #179. The `GuardDefinition` structure was modified to set the type of the `expectedValue` variable to `uint256`. All comparisons in the `_checkResult` function are

now performed between `uint256` values instead of `bytes32` values. The NatSpec documentation for the `GuardDefinition` structure now specifies the limitations of the valid guards in the system. The tests and example guards were modified to use the `uint256` type for the `expectedValue` variable.

TOB-SPL-13: Lack of contract existence checks on low-level calls
Resolved in PR #180. Contract existence checks were added to the `swap` function in the `Swapper` contract and the `runGuards` function in the `GuardManager` contract, as recommended. Some minor aesthetic changes were also introduced by the pull request.

TOB-SPL-14: Incorrect use of exchangeRates in doHardWork
Resolved in PR #176. The `exchangeRates` variable was replaced with the correct `assetGroupExchangeRates` variable. A new test was added to the "do hard work" integration test suite.

TOB-SPL-15: LinearAllocationProvider could return an incorrect result
Resolved in PR #177. The incorrect value in the `riskArray` variable was replaced with the correct one.

TOB-SPL-16: Incorrect formula used for adding/subtracting two yields
Resolved in PR #198 and PR #199. The total yield calculation formula was replaced with the correct one in the `Strategy`, `StrategyRegistry`, and `DepositManager` contracts. A new `MockStrategy` contract and a new integration test suite were added for the fee calculation. The Spool team also found an issue with the calculation of smart vault management fees. A resolution for this issue, along with the documentation of the issue, modifications to existing tests, and the addition of new tests, were submitted in pull request #199.

TOB-SPL-17: Smart vaults with re-registered strategies will not be usable
Resolved in PR #181. Removed strategies are now kept in a new mapping to prevent them from being re-added. An additional test was introduced in the `StrategyManager` suite to check for reverts when a previously removed strategy is added.

TOB-SPL-18: Incorrect handling of partially burned NFTs results in incorrect SVT balance calculation
Resolved in PR #182. A new `_simulateNFTBurn` function was added to take into account the correct amount of SVTs that will be obtained if the passed array of NFTs is burned. Test files for `SmartVaultManager` and integration tests were modified to match the new changes. This fix resolves both findings TOB-SPL-18 and TOB-SPL-19; some changes that the pull request makes in the files are related to finding TOB-SPL-19.

TOB-SPL-19: Transfers of D-NFTs result in double counting of SVT balance

Resolved in PR #182. The `_afterTokenTransfer` function and the `_activeUserNFTIds` and `_activeUserNFTCount` mappings were removed from the `SmartVault` contract, and checks were added in the `_beforeTokenTransfer` function to prevent self-transfers.

TOB-SPL-20: Flawed loop for syncing flushes results in higher management fees
Resolved in PR #185, PR #206, and PR #199. Checks for overlapping flush indexes and unsynced vaults were added to the flush process. These checks will make transactions revert if unsynced vaults are flushed. The fix for the fee calculation issue described in TOB-SPL-16 is also part of this issue resolution. The deposit integration tests were updated with the relevant cases.

TOB-SPL-21: Incorrect ghost strategy check
Resolved in PR #178. The check for the strategy role is now performed after the ghost strategy check in the `redeemStrategyShares` and `emergencyWithdraw` functions. Both functions now have a test that checks whether the ghost strategy is skipped in the withdrawal test suite.

TOB-SPL-22: Reward configuration not initialized properly when reward is zero
Partially resolved in PR #188. When a token is added, the `tokenAdded` configuration value is now initialized regardless of the reward parameter. A new test case was added to the `RewardManager` test suite to ensure that the `extendRewardEmission` function does not revert. However, the `periodFinish` configuration value is still not initialized when the reward is zero. This causes the `onlyFinished` modifier to revert and will require a manual call to the `extendRewardEmission` function before the `removeReward` function can be called.

TOB-SPL-23: Missing function for removing reward tokens from the blacklist
Resolved in PR #189. A new privileged function for the Spool admin role was added to the `RewardManager` contract to allow the removal of tokens from the blacklist, as recommended. Three additional test cases were added to the `RewardManager` test suite.

TOB-SPL-24: Risk of unclaimed shares due to loss of precision in reallocation operations
Resolved in PR #190. The total number of shares to be redeemed is now directly stored in the reallocation table. The `sharesToDistribute` value is now assigned directly in the `doReallocation` function, as recommended. A new test was added to the reallocation integration test file.

TOB-SPL-25: Curve3CoinPoolAdapter's _addLiquidity reverts due to incorrect amounts deposited
Resolved in PR #184. Rather than using an additional element at the end of the `assetsToDeposit` array to signal the need for a deposit, a separate Boolean variable is now used.

TOB-SPL-26: Reallocation process reverts when a ghost strategy is present
Resolved in PR #183. A check was added to the `mapStrategies` function to handle the presence of ghost strategies in the array of strategies, and a new test case was added to the reallocation test suite.

TOB-SPL-27: Broken test cases that hide security issues
Resolved in PR #205. All test cases indicated in the finding description were modified as recommended, along with numerous other test cases in different files. Additionally, as described in the other fix analyses in this section, new test cases were added or corrected during the process of fixing other issues. Even though it would be impossible to ensure that every situation has a test associated with it, the new test coverage is significantly better than the previous coverage.

TOB-SPL-28: Reward emission can be extended for a removed reward token
Resolved in PR #186. The `rewardConfiguration[smartVault][token]` element is now deleted when a reward is removed, as recommended. A new test for extending the reward emission after removing rewards was added.

TOB-SPL-29: A reward token cannot be added once it is removed from a smart vault
Resolved in PR #186. The fix applied for finding TOB-SPL-28 fixes this finding as well. The pull request also implements a test that removes a reward token, re-adds it, and verifies that the reward duration is correct.

TOB-SPL-30: Missing whenNotPaused modifier
Resolved in PR #187. The `whenNotPaused` modifier was added to the `claim` function in the `RewardPool` contract, as recommended. A new test was added to ensure the transaction reverts if the system is paused when `claim` is called.

TOB-SPL-31: Users who deposit and then withdraw before doHardWork lose their tokens
Resolved in PR #191. The `DepositManager` and `WithdrawalManager` contracts now have checks that verify the sync of the flush index and revert if the NFT is not synced. The `claimSmartVaultTokens` function in `DepositManager` and the `WithdrawalClaimBag` structure were modified to add the `flushIndexToSync` value. Relevant tests were added to the deposit and withdrawal integration test files.

TOB-SPL-32: Lack of events emitted for state-changing functions
Resolved in PR #202. Events were added for the state-changing functions indicated in the finding.

TOB-SPL-33: Removal of a strategy could result in loss of funds
Resolved in PR #197. When a strategy is removed, all funds that are waiting to be deposited, or to be synchronized in the case of withdrawals, are now sent to an emergency withdrawal wallet when the `_removeStrategy` function is called, and according to the Spool team, the assets will be distributed off-chain. The `doHardWork` and `claimWithdrawals` functions now track unclaimed assets. A total of nine tests were added or modified in the deposit and withdrawal integration tests.

TOB-SPL-34: ExponentialAllocationProvider reverts on strategies without risk scores
Resolved in PR #203. Allocation providers are now provided with an array of valid strategies, created by skipping the ghost strategy in the strategies array. Checks were added to verify the risk score values in the array of valid strategies and in the ghost strategy and to verify that the allocation sum adds up to a full percent. Relevant tests were added to the `RiskManager` and `SmartVaultFactory` test files.

TOB-SPL-35: Removing a strategy makes the smart vault unusable
Resolved in PR #192. The `flushFactors` array is now set to zero in the `calculateFlushFactors` function when a ghost strategy is found. The dust assignment code now checks that the strategy where the dust will be assigned is not a ghost strategy by checking that assets were already assigned to it. A new test was added to the deposit integration tests to check for deposits with ghost strategies.

TOB-SPL-36: Issues with the management of access control roles in deployment script
Resolved in PR #196. The README documentation file was updated with local test deployment instructions. Inline code documentation was added to describe the system's role permissions and the contracts that need specific roles. Missing role assignments were fixed. Tests for the `SmartVaultFactory` contract were updated to check ownership and role assignments. Major changes were made to the deployment script. New `JsonWriter` and `JsonReader` helper contracts are used to log and read configuration data.

TOB-SPL-37: Risk of DoS due to unbounded loops
Resolved in PR #194. The `GuardManager` and `ActionManager` contract now impose a hard limit of a maximum of 10 guards and actions, respectively. An additional check was added to `GuardManager` to prevent zero-length guards. The test suite was updated with new tests that cover the imposed limits.

TOB-SPL-38: Unsafe casts throughout the codebase
Resolved in PR #200. The unsafe casts in the `DepositManager` and `StrategyRegistry` contracts were modified to use OpenZeppelin's `SafeCast` library. Code comments were added in the functions to describe the assumptions made for the casts.

G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.