Zeebu Liquidation Adapter

Security Assessment

August 29, 2025

Prepared for:
Mrudul Chauhan
Zeebu

Prepared by: **Tarun Bansal**

Table of Contents

Table of Contents 1
Project Summary 2
Executive Summary 3
Project Goals 5
Project Targets 6
Project Coverage 7
Summary of Findings 8
Detailed Findings 9
1. The FlashLiquidationAdapterV3 contract does not return the remaining borrowed assets 9
2. The FlashLiquidationAdapterV3 contract cannot liquidate loans with the same collateral and borrowed assets 11
3. Front-running risks in the FlashLiquidationAdapterV3 contract 13
4. Lack of the initiator address validation 14
5. Integer underflow for insufficient flash loan amount 15
6. Insufficient slippage protection 17
7. Use of the incorrect swap function 19
A. Vulnerability Categories 21
B. Fix Review Results 23
Detailed Fix Review Results 25
C. Fix Review Status Categories 26
About Trail of Bits 27
Notices and Remarks 28

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Benjamin Samuels, Engineering Director, Blockchain
benjamin.samuels@trailofbits.com

The following consultants were associated with this project:

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Project Timeline

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

Date	Event
July 14, 2025	Pre-project kickoff call
July 17, 2025	Delivery of report draft
July 17, 2025	Report readout meeting
July 21, 2025	Delivery of final comprehensive report
August 6, 2025	Delivery of comprehensive report with fix review
August 29, 2025	Delivery of comprehensive report with second fix review

Executive Summary

Engagement Overview

Zeebu engaged Trail of Bits to review the security of their `FlashLiquidationAdapterV3` smart contract. The Zeebu money market protocol allows users to lend and borrow assets. The Zeebu team developed the `FlashLiquidationAdapterV3` smart contract to liquidate bad debt in the protocol using a flash loan and Uniswap V3 exchange.

A team of one consultant conducted the review from July 15 to July 16, 2025, for a total of two engineer-days of effort. Our testing efforts focused on testing the flash loan receiver, liquidation, and swap functionality. With full access to source code and documentation, we performed static and dynamic testing of the liquidation smart contract, using automated and manual processes. The money market pool contracts were out of scope for this review.

Observations and Impact

The `FlashLiquidationAdapterV3` smart contract is divided into small functions with limited responsibilities. The contract accepts user inputs in ABI-encoded data along with the parameters sent by the `Pool` contract's `flashLoan` function. It receives a flash loan, liquidates the specified user's bad debt with the flash loan, swaps the collateral asset earned from liquidation to the borrowed asset, and repays the flash loan.

During this review, we found that the contract uses an incorrect swap function to swap collateral assets to borrowed assets. This introduces several issues, including a medium-severity issue (TOB-ZEEBU-1), a low-severity issue (TOB-ZEEBU-6), and an issue of undetermined severity (TOB-ZEEBU-7). We found a denial-of-service (DoS) issue affecting bad loans using the same asset as collateral (TOB-ZEEBU-2). The lack of an access control check allows front-running, leading to another medium-severity issue (TOB-ZEEBU-3). Similarly, the lack of user input validation introduces two low-severity issues (TOB-ZEEBU-4 and TOB-ZEEBU-5).

Additionally, the test suite for the `FlashLiquidationAdapterV3` contract has several deficiencies. The test suite implements test cases for a mock Uniswap V3 router contract; some of these test cases test the same function, and some have the wrong description. The test suite does not test the correct functionality of the liquidation adapter, it does not test edge cases, and it does not implement negative test cases to test access control checks.

Finally, the contract has unused variables, `vars.remainingTokens` and `useEthPath`. The contract also includes debug log statements in the `executeOperation` function.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Zeebu take the following steps before deploying the contract:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Further divide the functions**. Consider dividing the `_liquidateAndSwap` function into two functions: one to handle the case of two separate collateral and borrowed assets, and another to handle the case of the same asset as both collateral and borrowed asset. This will simplify the logic and improve the contract's testability.
- **Improve the test suite.** We recommend that the Zeebu team define a clear testing strategy and create guidelines on how testing is performed in the codebase. The testing suite should have full coverage over all of the code paths in the contract with both positive and negative test cases. When implementing test cases, edge cases and malicious inputs should also be considered.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	3
Low	3
Informational	0
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Data Validation	3
Denial of Service	1
Timing	1
Undefined Behavior	1

Project Goals

The engagement was scoped to provide a security assessment of the Zeebu `FlashLiquidationAdapterV3` smart contract. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can an attacker cause a DoS on the liquidation adapter and stop the liquidation of bad loans?
- Can an attacker use the liquidation adapter to steal funds from the pool?
- Can an attacker steal funds from the liquidation adapter contract?
- Can funds get stuck in the liquidation adapter contract?
- Is the liquidation function implemented correctly?
- Are there any integer overflow/underflow issues?
- Does the contract validate all the user inputs?
- Are there any front-running risks?

Project Targets

The engagement involved reviewing and testing the following target.

Zeebu Liquidation Adapter

Repository	https://github.com/TechnologyZeebu/zeebu-v3-core
Version	`fff8315a99d375e54c1ea86f19e823c46388b8c5`
Directory	`zeebu-v3-code/contracts/flashloan`
Type	Solidity
Platform	EVM

Project Coverage

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

- **`FlashLiquidationAdapterV3`.** We reviewed the `FlashLiquidationAdapterV3` contract for common Solidity flaws, such as missing contract existence checks on low-level calls, issues with access controls, integer overflows, out-of-gas errors, reentrancy issues, and unimplemented functions. We also checked the `FlashLiquidationAdapterV3` contract for the correct implementation of the `IFlashLoanReceiver` interface. We reviewed the user input data decoding logic and validation logic. We assessed the implementation of the liquidation call and swap function to verify their correctness. We reviewed the access control checks to find issues leading to unauthorized access. Finally, we checked the contract for any logic bugs, loss of funds, and DoS vulnerabilities.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- The money market protocol contracts
- Compatibility with non-standard ERC20 tokens
- Deployment and configuration scripts
- Off-chain components interacting with the system

Summary of Findings

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

ID	Title	Type	Severity
1	The FlashLiquidationAdapterV3 contract does not return the remaining borrowed assets	Data Validation	**Medium**
2	The FlashLiquidationAdapterV3 contract cannot liquidate loans with the same collateral and borrowed assets	Denial of Service	**Medium**
3	Front-running risks in the FlashLiquidationAdapterV3 contract	Timing	**Medium**
4	Lack of the initiator address validation	Access Controls	**Low**
5	Integer underflow for insufficient flash loan amount	Data Validation	**Low**
6	Insufficient slippage protection	Data Validation	**Low**
7	Use of the incorrect swap function	Undefined Behavior	**Undetermined**

Detailed Findings

1. The FlashLiquidationAdapterV3 contract does not return the remaining borrowed assets

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ZEEBU-1
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description

The `FlashLiquidationAdapterV3` contract does not return the remaining borrowed assets to the `initiator` after repaying the flash loan.

The `FlashLiquidationAdapterV3` contract assumes that the Uniswap V3 router transfers exactly the amount required for paying back the flash loan. If the Uniswap router transfers a higher amount of borrowed assets, then the remaining amount is not transferred to the `initiator`. This remaining amount can be stolen by an attacker, as explained in the issue TOB-ZEEBU-4, or can remain stuck in the contract in the absence of a bad loan using the asset as collateral.

```
  ISwapRouter.ExactInputSingleParams memory params =
ISwapRouter.ExactInputSingleParams({
    tokenIn: collateralAsset,
    tokenOut: borrowedAsset,
    fee: 3000,
    recipient: address(this),
    deadline: block.timestamp,
    amountIn: vars.diffCollateralBalance,
    amountOutMinimum: vars.flashLoanDebt - vars.diffFlashBorrowedBalance,
    sqrtPriceLimitX96: 0
  });

  vars.soldAmount = SWAP_ROUTER.exactInputSingle(params);
  vars.remainingTokens = vars.diffCollateralBalance - vars.soldAmount;
} else {
  vars.remainingTokens = vars.diffCollateralBalance - premium;
}

IERC20(borrowedAsset).approve(address(POOL), vars.flashLoanDebt);
```

Figure 1.1: A snippet of the `_liquidateAndSwap` function
zeebu-v3-core/contracts/flashloan/adapters/FlashLiquidationAdapter.sol#L109–L126

Exploit Scenario

The Zeebu team liquidates a bad loan of 1,000 USDC by taking a flash loan of 1,000 USDC. The flash loan premium is 10 USDC, and the Uniswap swap router transfers 1,020 USDC for swapping the collateral asset earned from the liquidation. The `FlashLiquidationAdapterV3` contract pays the 1,100 USDC as flash loan repayment, but does not transfer the remaining 10 USDC to the `initiator`. An attacker can steal these 10 USDC.

Recommendations

Short term, transfer the remaining balance of the borrowed asset to the `initiator` after deducting the flash loan repayment amount.

Long term, consider asset price volatility when implementing smart contracts using decentralized exchanges to account for the total funds received from the exchange.

2. The FlashLiquidationAdapterV3 contract cannot liquidate loans with the same collateral and borrowed assets

Severity: **Medium**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-ZEEBU-2
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description

The `FlashLiquidationAdapterV3` contract cannot liquidate loans with the same collateral and borrowed assets.

The `FlashLiquidationAdapterV3` transfers the total balance of the collateral asset to the `initiator`, as shown in the figure below:

```
uint256 contractCollateralBalance =
IERC20(collateralAsset).balanceOf(address(this));
if (contractCollateralBalance > 0) {
  IERC20(collateralAsset).transfer(initiator, contractCollateralBalance);
}
```

Figure 2.1: A snippet of the `_liquidateAndSwap` function
zeebu-v3-core/contracts/flashloan/adapters/FlashLiquidationAdapter.sol#L128–L131

In case the collateral asset and borrowed asset are the same, this will transfer the total borrowed asset amount, including the amount required to pay back the flash loan, to the `initiator`. This causes the transaction to fail in the `flashLoan` function because the `Pool` contract cannot recover the flash loan amount from the `FlashLiquidationAdapterV3` contract. Therefore, the `FlashLiquidationAdapterV3` contract cannot be used to liquidate the loan using the same collateral and borrowed asset.

Exploit Scenario

Alice deposits 1,000 USDC in the money market pool and borrows 1,100 USDC by using her 1,000 USDC deposit as collateral. Alice does not pay the interest on the loan for the entire year, so her loan becomes unhealthy. The Zeebu team cannot liquidate this unhealthy loan because of the limitations of the `FlashLiquidationAdapterV3` contract.

Recommendations

Short term, transfer the collateral assets to the `initiator` only when the collateral asset and the borrowed asset differ. If both are the same, transfer only the remaining amount after deducting the flash loan repayment amount.

Long term, expand the test suite to implement test cases for all possible input values and edge cases.

3. Front-running risks in the FlashLiquidationAdapterV3 contract	
Severity: **Medium**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-ZEEBU-3
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description
An attacker can front-run the liquidator transactions to steal the funds transferred by the liquidator.

The Zeebu team uses the `FlashLiquidationAdapterV3` contract to liquidate bad loans in the money market. However, if the Zeebu team transfers the borrowed asset to the `FlashLiquidationAdapterV3` contract before calling the `flashLoan` on the `Pool` contract, then an attacker can front-run the `flashLoan` call to steal the funds from the `FlashLiquidationAdapterV3` contract by liquidating a bad loan. An attacker can keep repeating this attack to steal funds from the Zeebu team.

Exploit Scenario
The Zeebu team finds a bad loan of 1,000 USDC and notices that they can borrow 500 USDC from the pool as a flash loan. The Zeebu team transfers 500 USDC to the `FlashLiquidationAdapterV3` to liquidate the bad loan. The Zeebu team sends a transaction to liquidate the bad loan with a flash loan, but Eve front-runs this transaction and steals the 500 USDC from the `FlashLiquidationAdapterV3` by liquidating another bad loan.

Recommendations
Short term, implement a check in the `executeOperation` function of the `FlashLiquidationAdapterV3` contract to ensure that only trusted liquidators can use the liquidation adapter contract.

Long term, consider front-running risks while designing and implementing smart contracts.

4. Lack of the initiator address validation	
Severity: **Low**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-ZEEBU-4
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description

An attacker can steal leftover funds in the `FlashLiquidationAdapterV3` contract by using it for liquidation.

The `FlashLiquidationAdapterV3` contract does not validate the `initiator` of the flash loan and allows anyone to use it to liquidate a bad loan. Additionally, the `FlashLiquidationAdapterV3` contract transfers the leftover collateral asset to the `initiator`, as shown in the figure below:

```
uint256 contractCollateralBalance =
IERC20(collateralAsset).balanceOf(address(this));
if (contractCollateralBalance > 0) {
  IERC20(collateralAsset).transfer(initiator, contractCollateralBalance);
}
```

Figure 4.1: A snippet of the `_liquidateAndSwap` function
zeebu-v3-core/contracts/flashloan/adapters/FlashLiquidationAdapter.sol#L128–L131

Because leftover funds in the `FlashLiquidationAdapterV3` contract are transferred to the `initiator` address, anyone can steal these funds by using the contract for liquidation.

Exploit Scenario

Eve notices that the `FlashLiquidationAdapterV3` contract holds 10 WETH. She liquidates a bad debt of 1,000 USDC with a flash loan of 1,000 USDC to get WETH from the borrower's collateral. The `FlashLiquidationAdapterV3` contract transfers the 10 WETH balance along with the liquidation earnings to Eve.

Recommendations

Short term, store the address of the allowed liquidator, and validate the `initiator` to be only the allowed liquidator.

Long term, consider malicious behaviour when implementing access control checks for the system.

5. Integer underflow for insufficient flash loan amount

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ZEEBU-5
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description

The lack of validation of the flash loan amount against the `debtToCover` amount leads to gas fee loss from failed transactions.

The flash loan amount is not validated to be higher than the user-specified `debtToCover` amount. An insufficient flash loan amount will result in an integer underflow at the highlighted line in figure 5.1 and revert the transaction. The liquidator will lose the gas fee for executing the `flashLoan` and `liquidationCall` functions on the `Pool` contract.

```
if (collateralAsset != borrowedAsset) {
  vars.initFlashBorrowedBalance = IERC20(borrowedAsset).balanceOf(address(this));
  vars.borrowedAssetLeftovers = vars.initFlashBorrowedBalance - flashBorrowedAmount;
}

vars.flashLoanDebt = flashBorrowedAmount + premium;

IERC20(borrowedAsset).approve(address(POOL), debtToCover);
POOL.liquidationCall(collateralAsset, borrowedAsset, user, debtToCover, false);

uint256 collateralBalanceAfter = IERC20(collateralAsset).balanceOf(address(this));
vars.diffCollateralBalance = collateralBalanceAfter - vars.initCollateralBalance;

if (collateralAsset != borrowedAsset) {
  uint256 flashBorrowedAssetAfter = IERC20(borrowedAsset).balanceOf(address(this));
  vars.diffFlashBorrowedBalance = flashBorrowedAssetAfter -
vars.borrowedAssetLeftovers;
```

Figure 5.1: A snippet of the `_liquidateAndSwap` function
zeebu-v3-core/contracts/flashloan/adapters/FlashLiquidationAdapter.sol#L90–L106

Exploit Scenario

The `FlashLiquidationAdapterV3` holds 100 USDC. The Zeebu team notices a bad debt of 1,100 USDC and calls the `flashLoan` function on the `Pool` contract to borrow 1,000 USDC with the `FlashLiquidationAdapterV3` contract as the receiver. The `_liquidateAndSwap` function of the `FlashLiquidationAdapterV3` contract executes the `liquidationCall` function on the `Pool` contract successfully by transferring all 1,100

USDC to the `Pool` contract. However, the calculation of the `vars.diffFlashBorrowedBalance` fails because of an integer underflow while subtracting `vars.borrowedAssetLeftovers` (100) from `flashBorrowedAssetAfter` (0). The transaction reverts, and the liquidator loses the gas fee.

Recommendations

Short term, implement a check in the flash loan receiver contract to revert early if the flash loan amount is less than the `debtToCover` amount.

Long term, consider gas fee loss from reverted transactions and implement data validation checks to minimize this loss.

6. Insufficient slippage protection

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ZEEBU-6
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description

The Zeebu team can lose all liquidation profits to the MEV because of insufficient slippage protection in the Uniswap V3 router swap execution.

The `FlashLiquidationAdapterV3` contract swaps the collateral assets earned from liquidation to the borrowed assets to repay a flash loan. It computes the minimum amount of borrowed assets required to replay the flash loan and specifies this computed value as the `amountOutMinimum` parameter in the Uniswap V3 router swap function call:

```
ISwapRouter.ExactInputSingleParams memory params =
ISwapRouter.ExactInputSingleParams({
  tokenIn: collateralAsset,
  tokenOut: borrowedAsset,
  fee: 3000,
  recipient: address(this),
  deadline: block.timestamp,
  amountIn: vars.diffCollateralBalance,
  amountOutMinimum: vars.flashLoanDebt - vars.diffFlashBorrowedBalance,
  sqrtPriceLimitX96: 0
});
```

Figure 6.1: A snippet of the `_liquidateAndSwap` function
zeebu-v3-core/contracts/flashloan/adapters/FlashLiquidationAdapter.sol#L109–L118

However, if the collateral assets earned from liquidation are worth more than the minimum required borrowed assets, an attacker can sandwich the liquidation transaction to increase the swap slippage for the liquidator and profit from it. The liquidator can lose all liquidation profits to the attacker because of the insufficient slippage protection in the swap function call.

Exploit Scenario

Eve monitors the mempool for liquidation transactions and sandwiches them to profit from it at the cost of the liquidator.

Recommendations

Short term, compute the minimum amount out by querying the price from the Uniswap SDK or an on-chain price oracle, and use that value in the swap function call.

Long term, review the Uniswap documentation for single swaps along with the references below to understand slippage and the risks of using decentralized exchanges.

References

- DeFi Sandwich Attacks
- Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges

7. Use of the incorrect swap function

Severity: **Undetermined**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-ZEEBU-7
Target: `contracts/flashloan/adapters/FlashLiquidationAdapter.sol`	

Description

The `_liquidateAndSwap` function swaps the collateral asset for the borrowed asset using the `exactInputSingle` function instead of the `exactOutputSingle` function, which leads to an incorrect remaining balance computation.

The `FlashLiquidationAdapterV3` contract swaps the collateral assets earned from liquidation to the borrowed asset to repay the flash loan. However, it swaps using the `exactInputSingle` function instead of the `exactOutputSingle` function. The `exactInputSingle` function returns the amount of borrowed assets transferred in a swap; this amount is then subtracted from the collateral asset balance to compute the remaining amount of the collateral asset tokens. This design is incorrect and leads to unexpected behaviour.

```
ISwapRouter.ExactInputSingleParams memory params =
ISwapRouter.ExactInputSingleParams({
  tokenIn: collateralAsset,
  tokenOut: borrowedAsset,
  fee: 3000,
  recipient: address(this),
  deadline: block.timestamp,
  amountIn: vars.diffCollateralBalance,
  amountOutMinimum: vars.flashLoanDebt - vars.diffFlashBorrowedBalance,
  sqrtPriceLimitX96: 0
});

vars.soldAmount = SWAP_ROUTER.exactInputSingle(params);
vars.remainingTokens = vars.diffCollateralBalance - vars.soldAmount;
```

Figure 7.1: A snippet of the `_liquidateAndSwap` function
zeebu-v3-core/contracts/flashloan/adapters/FlashLiquidationAdapter.sol#L109–L121

Recommendations

Short term, use the `exactOutputSingle` swap function of the Uniswap V3 router to swap the tokens.

Long term, improve unit test cases to test the system behaviour. Check the user's and system contract's balance after the transaction to ensure that the assets are transferred correctly.

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. 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.

On July 28, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Zeebu team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

During our August 28, 2025 review of the TOB-ZEEBU-3 fix, we identified some additional considerations that emerged from the code changes made to address the original issues. The initial commit #bb44cbf introduced some unintended side effects that temporarily impacted contract usability. Commit #3965977, delivered on August 29, 2025, successfully resolved both the original issues and the subsequent items we flagged.

This review process has highlighted an opportunity to strengthen the testing framework. Currently, the test suite relies on mock contracts for the flash loan protocol, swap protocol, and ERC20 token transfers, which may not fully capture the nuanced behaviors of the actual implementations. We recommend considering the addition of integration tests and fork test suites to provide more comprehensive coverage of the liquidation adapter contract's functionality in realistic environments. This enhancement would help catch potential issues earlier in the development cycle and reduce the need for multiple fix iterations.

In summary, of the seven issues described in this report, Zeebu has resolved all seven issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	The FlashLiquidationAdapterV3 contract does not return the remaining borrowed assets	Medium	Resolved
2	The FlashLiquidationAdapterV3 contract cannot liquidate loans with the same collateral and borrowed assets	Medium	Resolved
3	Front-running risks in the FlashLiquidationAdapterV3 contract	Medium	Resolved
4	Lack of the initiator address validation	Low	Resolved

5	Integer underflow for insufficient flash loan amount	Low	Resolved
6	Insufficient slippage protection	Low	Resolved
7	Use of the incorrect swap function	Undetermined	Resolved

Detailed Fix Review Results

TOB-ZEEBU-1: The FlashLiquidationAdapterV3 contract does not return the remaining borrowed assets
Resolved in PR#3. The remainder (if any) of repaying the flash loan is now transferred back to the initiator.

TOB-ZEEBU-2: The FlashLiquidationAdapterV3 contract cannot liquidate loans with the same collateral and borrowed assets
Resolved in PR#3. The flash loan repayment now correctly handles the case in which both assets are the same, transferring back only the remainder after repayment (if any).

TOB-ZEEBU-3: Front-running risks in the FlashLiquidationAdapterV3 contract
Resolved in commits #bb44cbf and #3965977. The `executeOperation` function now computes the already-existing asset amounts and deducts them from the asset amounts to transfer to the `initiator` at the end of the liquidation process. This prevents transfer of existing assets to the `initiator`, thus preventing the theft of assets by front-running.

TOB-ZEEBU-4: Lack of the initiator address validation
Resolved in PR#3. There are no longer leftover funds in the adapter after a liquidation.

TOB-ZEEBU-5: Integer underflow for insufficient flash loan amount
Resolved in PR#3. An additional check was included to enforce `flashBorrowedAmount` to be higher than or equal to `debtToCover`.

TOB-ZEEBU-6: Insufficient slippage protection
Resolved in PR#3. Callers now establish their own slippage protection instead of calculating it at the contract level.

TOB-ZEEBU-7: Use of the incorrect swap function
Resolved in PR#3. The `exactInputSingle` function is still in use; however, its output is ignored, and instead, the adapter's balance is checked after the call to calculate the swap result.

C. 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.

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 assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up to date with our latest news and announcements, please follow @trailofbits on X or LinkedIn, 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

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information 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 software controls and security properties. 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. A project's time and resource constraints also limit their use.