minterest.

SMART CONTRACT AUDIT

zokyo

October 18th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING MINTEREST

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- Medium issues: 0 points deducted
- Low issues: 2 issues (1 fixed, 1 verified) = 0 points deducted
- Informational issues: 3 issues (all verified) = -2 points deducted as 2 issues while being verified still raise concerns from auditor's side

And also 4 point deducted due to centralization risks and dependency on the out-of-scope service

Thus, 100 - 2 - 4 = 94

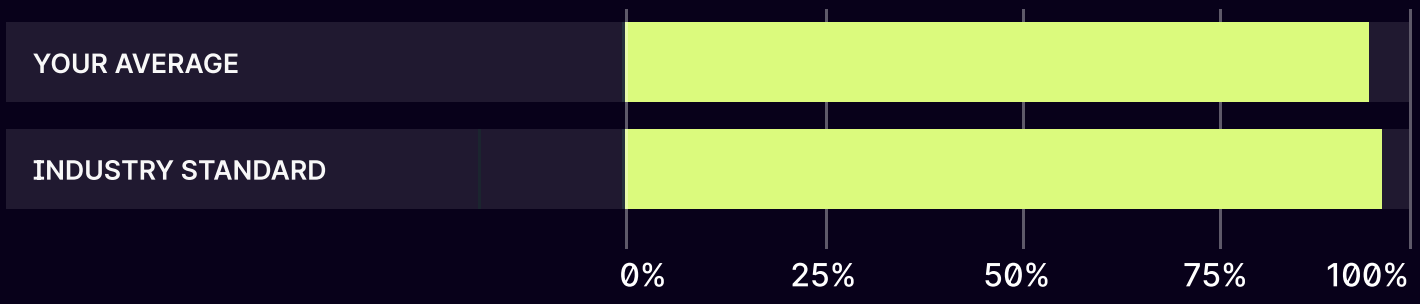
TECHNICAL SUMMARY

This document outlines the overall security of the Minterest smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Minterest smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



Auditors verified the code to be testable enough, to match the industry standard.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Minterest team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Minterest repository:

Initial commit: `b02201b88e16c025f238d0a1211073a42a0dc1d8`
Final commit: `007caac4e3af041c5db9fc1784adb1fc809125a3` (PR commit)

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- Flasher.sol
- Liquidation.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Minterest smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

Zokyo Security received a set of contracts from the Minterest team, which included an updated liquidation contract and a new flasher contract. The Minterest protocol had been audited before, as had the main contracts in the last iteration.

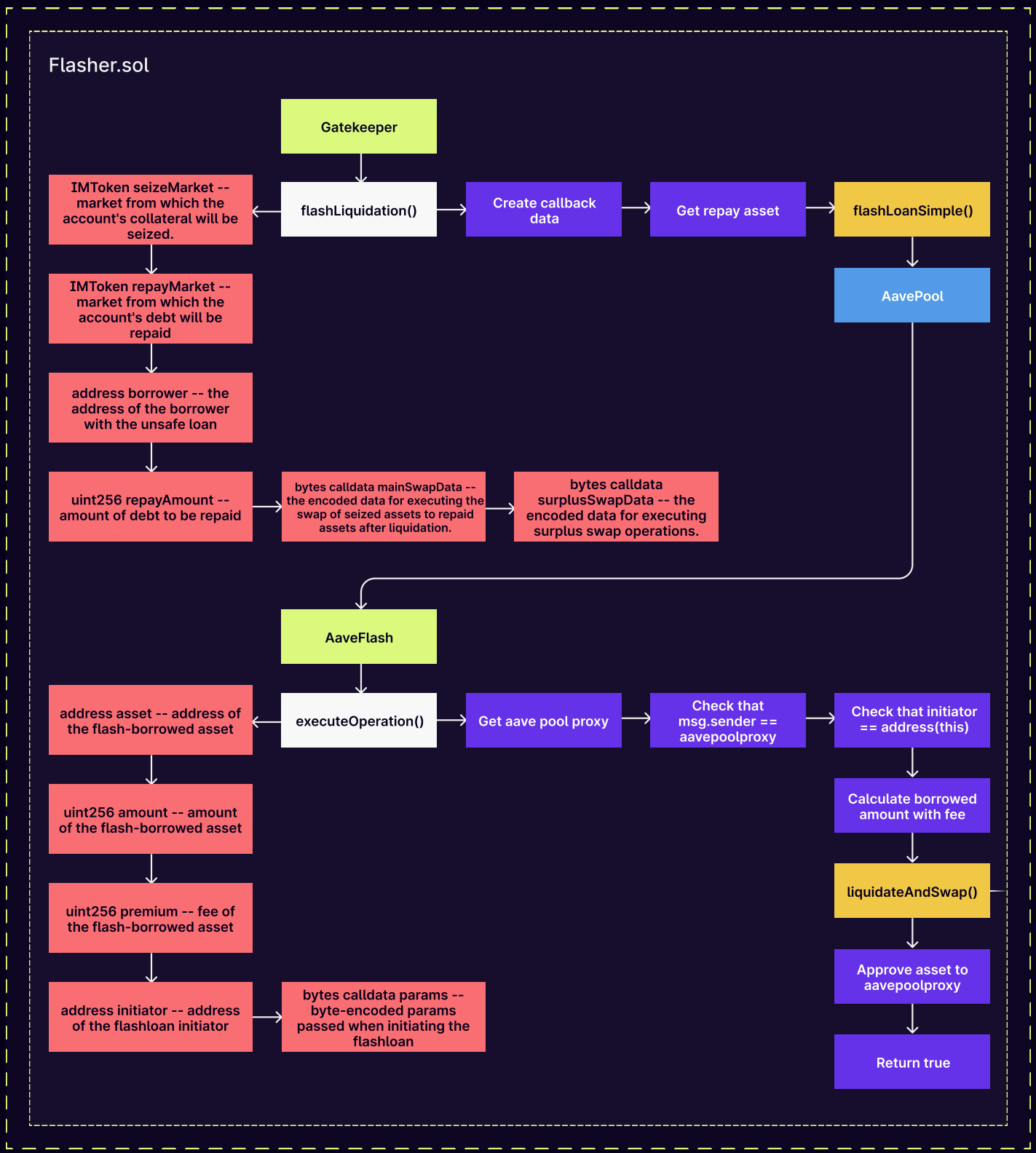
The audit's goal was to verify the updated functionality and ensure that the new contract functioned correctly and had a known level of security. The auditors checked the code line by line, compared it against their own checklist of vulnerabilities, validated the business logic of the contracts, and ensured that best practices in terms of gas spending were applied. The setup and deployment scripts of the contracts were also carefully audited.

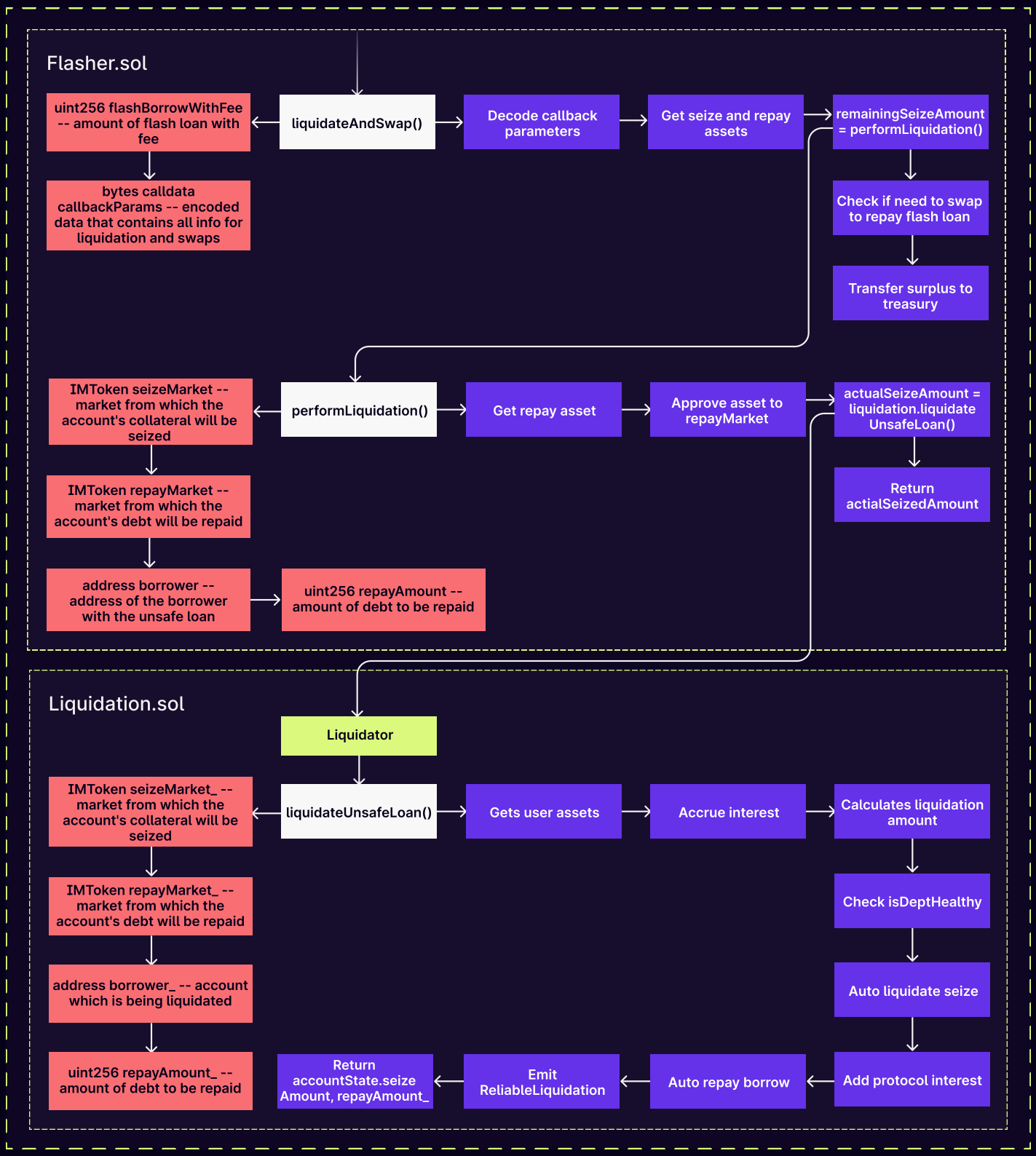
During the manual part of the audit, only a few low-level and informational-severity issues were found. Issues were related to a lack of events, code readability, and loop iteration problems. The Minterest team resolved most of them, while others were discussed and verified.

Another part of the audit process involved checking the native tests prepared by the Minterest team and preparing a set of custom test scenarios such as different health factor liquidations. It should be noted that the Minterest team provided a solid set of tests covering all the contracts' core logic. Nevertheless, Zokyo Security prepared its own set of unit tests and additional scenarios to cover the complex functionality of Minterest contracts. The complete set of unit tests can be seen in the Code Coverage and Test Result sections.

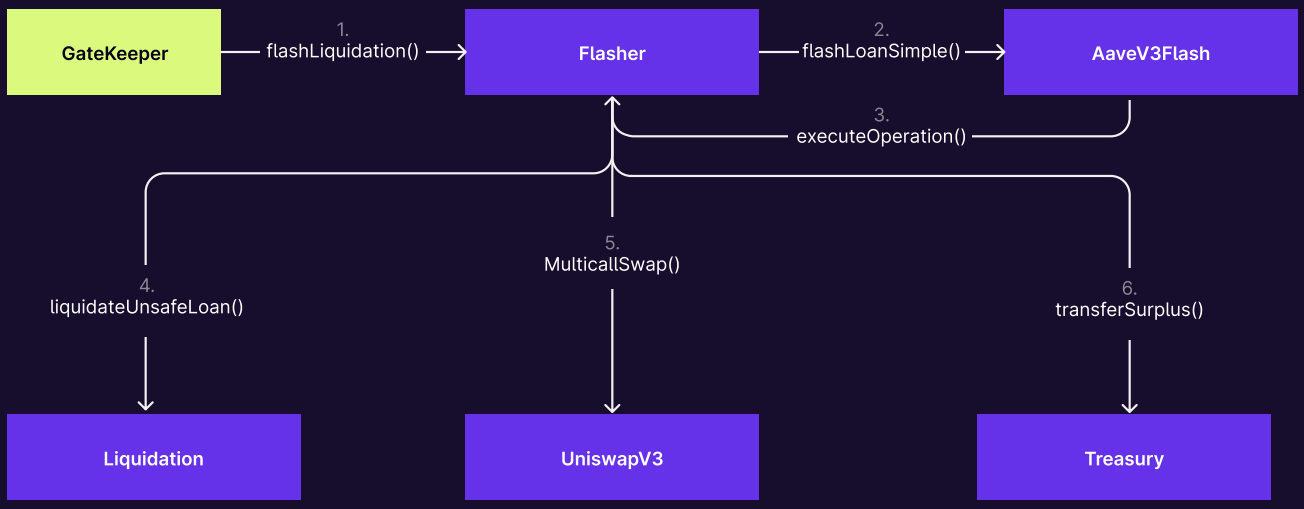
Overall, the security level of the contracts is high. The contracts are well-written and tested, with an updated version of the Liquidation contract, and a new Flasher contract being carefully audited. Though, auditor's team needs to notice, that contracts are dependent on the backend service and healthy behaviour of the admin and safe storage of private keys.

Protocol overview

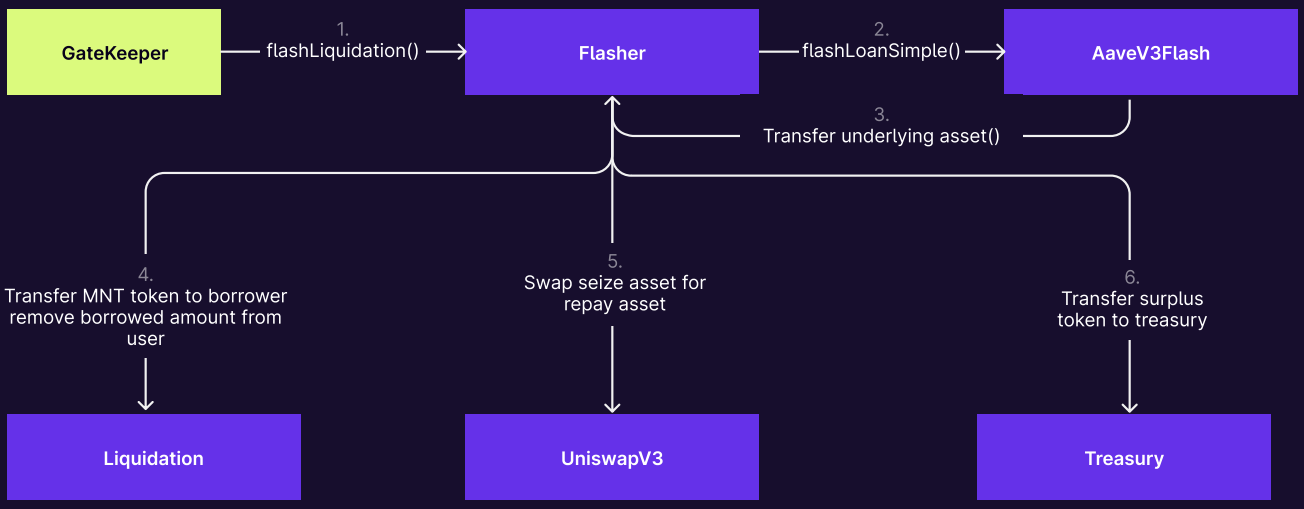



Flasher.sol
uint256 flashBorrowWithFee -- amount of flash loan with fee
liquidateAndSwap()
Decode callback parameters
Get seize and repay assets
remainingSeizeAmount = performLiquidation()
bytes calldata callbackParams -- encoded data that contains all info for liquidation and swaps
Check if need to swap to repay flash loan
Transfer surplus to treasury
IMToken seizeMarket -- market from which the account's collateral will be seized
performLiquidation()
Get repay asset
Approve asset to repayMarket
actualSeizeAmount = liquidation.liquidate UnsafeLoan()
IMToken repayMarket -- market from which the account's debt will be repaid
Return actialSeizedAmount
address borrower -- address of the borrower with the unsafe loan
uint256 repayAmount -- amount of debt to be repaid
Liquidation.sol
Liquidator
IMToken seizeMarket_ -- market from which the account's collateral will be seized
liquidateUnsafeLoan()
Gets user assets
Accrue interest
Calculates liquidation amount
IMToken repayMarket_ -- market from which the account's debt will be repaid
Check isDeptHealthy
address borrower_ -- account which is being liquidated
Auto liquidate seize
uint256 repayAmount_ -- amount of debt to be repaid
Return accountState.seize Amount, repayAmount_
Emit ReliableLiquidation
Auto repay borrow
Add protocol interest

Flash liquidation flow



Flash liquidation money flow



Roles and Responsibilities

Flasher.sol

1. Deployer

The deployer is responsible for the deployment of Flasher.sol. The deployer must pass valid initial parameters such as the admin is responsible for address, aave pool address, swap router address, liquidation contract address, price oracle address, default surplus asset address and treasury address.
The admin becomes the admin of the contract and gets the gatekeeper role during deployment.

2. Admin

The Admin is responsible for withdrawal assets from the contract, setter for tokenOutDeviation, update contract addresses, add/remove allowed surplus assets, as well as receivers and gatekeeper.

Functions called by admin:

- withdraw()
- setTokenOutDeviation()
- setRouterAddress()
- setLiquidationAddress()
- setPoolAddressesProvider()
- setOracleAddress()
- setTreasuryAddress()
- setDefaultSurplusAsset()
- addAllowedReceiver()
- addAllowedSurplusAsset()
- addAllowedGatekeeper()
- removeAllowedSurplusAsset()
- removeAllowedReceiver()
- removeAllowedGatekeeper()

3. Gatekeeper

The gatekeeper is responsible for flash liquidations which is used to liquidate unsafe loan and transfer surplus asset to the treasury. The gatekeeper should validate calldata for flash liquidation such as mainSwapData and surplusSwapData to correctly invoke function.

Functions that can be called by the gatekeeper:

- flashLiquidation()

MINTEREST DESCRIPTION

Liquidation.sol

1. Deployer

The deployer is responsible for the deployment of Liquidation.sol. The deployer must pass valid initial parameters such as list of trusted liquidators address, supervisor contract address and admin address. The admin becomes the admin of the contract and gets the trusted_liquidator and timelock roles during the deployment.

2. Admin

The admin role has no functionality in this contract.

3. The Trusted Liquidator

The trusted liquidator is responsible for liquidate unsafe loans.

Functions that can be called by the trusted liquidator:

- liquidateUnsafeLoan()

4. Timelock

The timelock role is responsible for setting new healthy factor limit.

Functions that can be called by the trusted timelock:

- setHealthyFactorLimit

Deployment script

Flasher.sol

Deployment script is located at deploy/scripts/ProtocolProduction/flasher.js

Path for addresses that needed for deployment located at configGetters.js file.

After deployment script check for correct setting for gatekeeper role, liquidation address, oracle address, aavePool address, swap router address, defaultSurplusAsset address and treasury address. Contract verifies using verify() function from verify.js script.

Liquidation.sol

Deployment script is located at deploy/scripts/ProtocolProduction/liquidation.js

Path for addresses that needed for deployment located at configGetters.js file.

After deployment script check for correct setting for liquidation role, supervisor address and healthyFactorLimit. Contract verifies using verify() function from verify.js script.

MINTEREST DESCRIPTION

List of valuable assets

Flasher.sol

1. **Underlying asset: token that can be stored in contract.**

Supervisor.sol

1. **accountAssets: assets that can be liquidated for user.**

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

LOW-1 | VERIFIED

Lack of events.

Flasher.sol: flashLiquidation(), executeOperation()
The events from the functions above are not emitted. However, they can keep information on important operations on the contract, so these operations can be tracked in the future.

Recommendation:
Emit events in the given functions.

Post-audit:
Minterest team verified that given transactions can be verified by events from FlashLoan, ReliableLiquidation, MulticallSwap, etc.

LOW-2 | RESOLVED

Remaining seize amount is not validated.

Flasher.sol: liquidateAndSwap(), line 302.
The variable `remainingSeizeAmount` which represents the leftover of seize token after paying the flashloan fee is not validated not to be equal to 0. As a result, the exchange of seize asset into surplus asset may revert and prevent liquidation. Thus, it is recommended to avoid execution of functions exactInSingleSwap() and transferSurplus() in case `remainingSeizeAmount` equals zero.

Recommendation:
Validate if `remainingSeizeAmount` equals 0.

Post-audit:
Added validation for `remainingSeizeAmount`.

INFORMATIONAL-1 | VERIFIED

Underscore Prefix for Non-external Functions and Variables.

All contracts.
It is recommended to use underscore prefix to identify non-external or non-public functions and variables as it will improve code readability. Source: https://docs.soliditylang.org/en/latest/style-guide.html#underscore-prefix-for-non-external-functions-and-variables

Recommendation:
Use underscore at the start of function names in non-external functions/variables.

Post-audit:
Minterest team verified that issue will not be fixed in current iteration.

INFORMATIONAL-2 | VERIFIED

Verify validation of swap data.

Flasher.sol: flashLiquidation(), arguments mainSwapData, surplusSwapData
As calldata information should provide backend service that is not in the audit scope, verify that information for flashLiquidation() is validated before invoking the function. This issue was marked as info since the necessary validations are added in the validateAmountsAndNullifyAllowance(), however validating the call datas is helpful to avoid reverts.

Recommendation:
Verify that arguments are validated by backend before sending the transaction.

Post-audit:
Minterest team verified that calldata is provided by their own backend service and final results are validated in validateAmountsAndNullifyAllowance(). Additional validation of input calldata will not give any additional security improvement, but will significantly increase gas. Though, despite the verification, auditors team still leave a concern since the actual data and backend service are out of the scope of the audit.

Iteration through users' open market may run out of gas.

Liquidation.sol: liquidateUnsafeLoan(), supervisor.getAccountAssets() → calculateLiquidationAmounts(), lines 150-209.
As user can have multiple asset markets, if he has too many of them, when liquidate loan will be invoked, the transaction could potentially run out of gas as many iterations with different markets can consume all gas in transaction. Consider adding a limitation for markets that a user can have in the Supervisor contract.

Recommendation:
Add markets limit for user to prevent transactions run out of gas OR verify that open markets max number will be optimal, so that even if the user opens all of them, the transaction will not run out of gas.

Post-audit:
Minterest team verified that they will never use such an amount of markets, that this becomes an actual issue. Adding new markets is limited by business conditions - they need deep DEX pools and flash loan source for all assets. Though, despite the verification, auditors team still leave a concern regarding such behavior.

	Flasher.sol Liquidation.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Minterest in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Minterest contract requirements for details about issuance amounts and how the system handles these.

Flasher

Liquidate using flash loan

✓ flashLiquidation WETH -> WETH (277ms)

Admin functions tests

✓ Withdraw (54ms)
✓ Set token out deviation (210ms)
✓ Set new router address (172ms)
✓ Set new liquidation address
✓ Set new pool provider address
✓ Set new oracle address
✓ Set new treasury address
✓ Set new default surplus address
✓ Add allowed receiver address
✓ Add allowed surplus address
✓ Add allowed gatekeeper address
✓ Remove allowed receiver address
✓ Remove allowed surplus address
✓ Remove allowed gatekeeper address

Revert

✓ If not admit tries to withdraw (47ms)
✓ If try to withdraw for not allowed receiver
✓ If try to withdraw more amount that on contract
✓ If not admit tries to set token out deviation
✓ If try to set token out deviation to zero
✓ If not admit tries to set new router address
✓ If try to set new router address to zero
✓ If not admit tries to set new liquidation address
✓ If try to set new liquidation address to zero
✓ If not admit tries to set new pool provider address (48ms)
✓ If try to set new pool provider address to zero

✓ If not admit tries to set new oracle address (44ms)
✓ If try to set new oracle address to zero
✓ If not admit tries to set new treasury address (39ms)
✓ If try to set new treasury address to zero
✓ If not admit tries to set new default surplus address
✓ If try to set new default surplus address to zero
✓ If not admit tries to add allowed receiver address (53ms)
✓ If try to add allowed receiver address to zero
✓ If not admit tries to add allowed surplus address
✓ If try to allowed surplus address to zero
✓ If not admit tries to add allowed gatekeeper address
✓ If try to allowed gatekeeper address to zero
✓ If not admit tries to remove allowed receiver address
✓ If not admit tries to remove allowed surplus address
✓ If not admit tries to remove allowed gatekeeper address

41 passing (3s)

FILE	% STMTS	% BRANCH	% FUNCS
Flasher.sol	88.42	76.56	96.43
Liquidation.sol	94.92	65.22	87.5
All Files			

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Minterest

As a part of our work assisting Minterest in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the Minterest team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the Minterest team. All of them were also carefully checked by the team of auditors.

Flash Loans

maxFlashLoan

✓ returns share of balance
✓ returns zero on other tokens

flashFee

✓ returns share of amount
✓ wrong token arg reverts

change params

✓ setFlashLoanMaxShare
✓ setFlashLoanFeeShare
✓ should revert if args are greater than 1e18 (42ms)
✓ should be timelock only (40ms)

flashLoan

✓ should call callback (53ms)
✓ should transfer tokens (51ms)
✓ there should be tokens inside callback (230ms)
✓ should increase protocol interest by fee (53ms)
✓ should return true (42ms)
✓ should accrue market interest (54ms)
✓ should revert if is paused in supervisor (51ms)
✓ should revert if not enough fee (62ms)
✓ should revert if not enough allowance (45ms)
✓ should revert if token is not underlying
✓ should revert if asks more than max
✓ should revert if callback returns not success

Flasher contract

Admin functions tests

✓ withdraw should work
✓ withdraw should revert if called by not-admin
✓ withdraw should revert if receiver is not in the whitelist

✓ withdraw should revert if receiver there is not enough liquidity
✓ setRouterAddress should work
✓ setRouterAddress should revert if called by not-admin
✓ setRouterAddress should revert if called with zero address arg
✓ setPoolAddressesProvider should work
✓ setPoolAddressesProvider should revert if called by not-admin
✓ setPoolAddressesProvider should revert if called with zero address arg
✓ setOracleAddress should work
✓ setOracleAddress should revert if called by not-admin
✓ setOracleAddress should revert if called with zero address arg
✓ setTreasuryAddress should work
✓ setTreasuryAddress should revert if called by not-admin
✓ setTreasuryAddress should revert if called with zero address arg
✓ setDefaultSurplusAsset should work
✓ setDefaultSurplusAsset should revert if called by not-admin
✓ setDefaultSurplusAsset should revert if called with zero address arg
✓ setLiquidationAddress should work
✓ setLiquidationAddress should revert if called by not-admin
✓ setLiquidationAddress should revert if called with zero address arg
✓ addAllowedReceiver should work
✓ addAllowedReceiver should revert if called by not-admin
✓ addAllowedReceiver should revert if called with zero address args
✓ addAllowedSurplusAsset should work
✓ addAllowedSurplusAsset should revert if called by not-admin
✓ addAllowedSurplusAsset should revert if called with zero address args
✓ addAllowedGatekeeper should work
✓ addAllowedGatekeeper should revert if called by not-admin
✓ addAllowedGatekeeper should revert if called with zero address args
✓ removeAllowedReceiver should work
✓ removeAllowedReceiver should revert if called by not-admin
✓ removeAllowedSurplusAsset should work
✓ removeAllowedSurplusAsset should revert if called by not-admin
✓ removeAllowedGatekeeper should work
✓ removeAllowedGatekeeper should revert if called by not-admin
✓ setTokenOutDeviation should work
✓ setTokenOutDeviation should revert if called by not-admin
✓ setTokenOutDeviation should revert if newValue == 0
validateAmountsAndNullifyAllowance tests
✓ validateAmountsAndNullifyAllowance should not throw if amounts are in range for exactIn trade type
✓ validateAmountsAndNullifyAllowance should not throw if amounts are in range for exactOut trade type
✓ validateAmountsAndNullifyAllowance should revert if invalid amount tokenIn spent, exactIn trade type

✓ validateAmountsAndNullifyAllowance should revert if invalid amount tokenOut recieved, exactIn trade type
✓ validateAmountsAndNullifyAllowance should revert if allowance is incorrect, exactIn trade type
✓ validateAmountsAndNullifyAllowance should revert if invalid amount tokenIn spent, exactOut trade type
✓ validateAmountsAndNullifyAllowance should revert if invalid amount tokenOut recieved, exactOut trade type
✓ validateAmountsAndNullifyAllowance should work correctly with tokenOutDeviation, exactOut trade type

Internal functions

✓ isSurplusSwapRequired should work
✓ transferSurplus should work

calculateAmountOutMinimum

✓ Should calculate AmountOutMinimum correctly

should revert in case of invalid price

✓ price for TokenIn is zero
✓ price for TokenOut is zero

Liquidation contract

✓ Check initial params
✓ Should revert: supervisor address can`t be address zero (175ms)
✓ Should revert: admin address can`t be address zero
✓ setHealthyFactorLimit() should work
✓ setHealthyFactorLimit() should fail (42ms)
✓ oracle() should work correctly (508ms)

approveBorrowerHealthyFactor()

✓ should return true if currentHealthyFactor is correct
✓ should fail if zero params were passed

accrue()

✓ Should work correctly
✓ Should fail if incorrect market address passed

calculateLiquidationAmounts()

✓ Fails if asset price is 0
✓ Fails if asset price causes overflow
✓ Fails if the account borrow underlying causes overflow
✓ Fails if getAccountSnapshot() causes overflow
✓ Fails if repay amount is GT borrow amount
✓ Fail if selected collateral markets don't cover required seizeAmount (49ms)
✓ Should returns correct values (48ms)
✓ Should calculates only total values If seizeMarket correct BUT repayMarket is incorrect (45ms)
✓ Should calculate only total values If seizeMarket && repayMarket are correct BUT repayAmount is zero (47ms)
✓ Should calculate only total values If seize && repay markets are incorrect (BT service

usage case) (49ms)
✓ Should fail If repayMarket is correct && repayAmount is NOT zero BUT seizeMarket is incorrect (46ms)
✓ Should work correctly if UF for market is zero (54ms)
✓ Should work correctly if LF for market is zero (55ms)
LiquidateUnsafeLoan() should work correctly
✓ Have to be reverted with access control error
✓ Have to be reverted with Insufficient shortfall (38ms)
✓ Have to be reverted with Healthy factor not in range (93ms)

99 passing (21s)

FILE	% STMTS	% BRANCH	% FUNCS
Flasher.sol	57.89	82.81	75
Liquidation.sol	91.53	84.78	100
All Files	**74.71**	**83.8**	**87.5**

We are grateful for the opportunity to work with the Minterest team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Minterest team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo