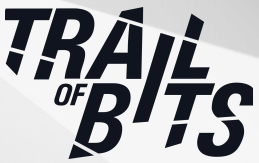

Cod3x Foundation

Cod3x Lend

Security Assessment

June 18, 2024

Prepared for:
Justin Bebis

Cod3x Foundation

Prepared by: **Gustavo Grieco and Damilola Edwards**

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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
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

Project Summary

Contact Information

The following project manager was associated with this project:

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

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Damilola Edwards, Consultant
damilola.edwards@trailofbits.com

Project Timeline

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

Date	Event
March 7, 2024	Pre-project kickoff call
March 15, 2024	Status update meeting #1
March 22, 2024	Delivery of report draft
March 22, 2024	Report readout meeting
June 18, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

The Byte Masons Cod3x Foundation engaged Trail of Bits to review the security of Cod3x Lend, a liquidity market that allows lending assets to other sub-markets and also rehypothecates the underlying collateral assets.

A team of two consultants conducted the review from March 11 to March 22, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on uncovering improper arithmetic related to user funds, undefined behavior, access control issues, and denial-of-service (DoS) conditions. With full access to source code and documentation, we performed static analysis of the provided codebase and conducted a manual code review.

Observations and Impact

During this engagement, we uncovered two high-severity findings affecting the interactions between the lending pool and the mini pool contracts. One of these was the lack of access controls, which could potentially allow an attacker to evade liquidation indefinitely (TOB-GRANARY-006).

In addition, we discovered issues with the debt accounting in the mini pools, particularly concerning internal accounting of assets borrowed from the lending pool. The current implementation fails to decrease its debt value when the debt is repaid, only increasing it when assets are borrowed (TOB-GRANARY-007). This fragile accounting system also makes the mini pools susceptible to donation attacks that may lead to a DoS when repaying debt to the lending pool.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that the Cod3x Lend team take the following steps prior to deployment:

- **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**.**
- **Incorporate Slither into the CI pipeline.** Slither can help the development team proactively address informational-level concerns and align the protocol with best practices.
- **Expand the test suite to cover edge cases.** Improve the existing unit tests and expand fuzz tests to cover edge conditions, especially mathematical operations, access controls, and accounting logic.

- **Consider another security review.** The code is still a work in progress and contains several high-impact flaws, so another code review is necessary to verify its security properties.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	6
Low	0
Informational	1
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Access Control	2
Auditing and Logging	1
Data Validation	2
Undefined Behavior	5

Project Goals

The engagement was scoped to provide a security assessment of Byte Masons' Cod3x Lend. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can protocol funds be stolen?
- Can any part of the protocol's functionality be disabled?
- Are there any deficiencies in the math used by the protocol that would result in a loss of funds or other problems?
- Can funds be locked or lost in any way?
- Are there any potential DoS attacks?
- Does the protocol handle external interactions safely?
- Are all inputs properly validated before being used? Do the contracts correctly handle both expected and unexpected inputs?

Project Targets

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

Granary-v2

Repository	https://github.com/The-Granary/granary-v2
Version	`1388027cd87a7bff2a0b13628dc23ba71128bec1`
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 included the following:

- Lending pool changes: The codebase is a fork of Aave V2 with some changes introduced to the lending pool logic to facilitate interactions with the mini pools. We reviewed these changes, which include new operations and specific interactions with the mini pools in order to borrow and repay assets when necessary.

- Mini pool: The mini pool is a submarket that can borrow from the main lending pool. We reviewed the interactions between users and mini pools. In particular, we reviewed how users deposit, borrow, and repay assets. We looked for cases where users can disrupt or break mini pools' internal bookkeeping or expected invariants. Additionally, we verified that mini pools' positions cannot be liquidated in the lending pool.

- Rehypothecation: when aTokens are burned or minted, the contract performs a rebalance to free funds for future transactions. We reviewed how and when rehypothecation is performed. Additionally, we reviewed interactions with the ERC-4626 contract during this process to make sure they follow the specification.

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:

- Flash loan code was out of scope for this engagement.

- Economic-related attacks to the components were not considered; instead, the review focused on low-level attacks.

- The mini pool interest rate computation was superficially reviewed and needs to be documented and extensively tested to provide good security guarantees.

- We did not perform any dynamic testing such as fuzzing.

- Any third-party component used directly or indirectly was out of scope—for instance Solady's ERC-6909 and OpenZeppelin contracts.

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	Most of the arithmetic-related complexity is located in the interaction between the lending pools and the mini pools, in the rehypothecation, and in the interest rate computation. These features need extensive documentation, including how parameters are bounded, what invariants are important, and how the invariants are enforced. Additionally, while unit tests cover most of the system's arithmetic, the use of a fuzzer—or a similar automated testing method—is still a work in progress. Using a fuzzer would increase confidence in the arithmetic operations.	**Moderate**
Auditing	For the most part, sufficient events are emitted during critical operations to facilitate off-chain monitoring, following the AAVE 2.0 codebase and practices. However, events associated with rehypothecation are missing (TOB-GRANARY-009), which reflects a lack of testing for proper event emission. Details regarding the incident response and off-chain monitoring plans are unclear.	**Moderate**
Authentication / Access Controls	The roles and responsibilities of all actors should be clarified. In particular, documentation is needed to explain how mini pools interact with the lending pool. We found major access control issues, including TOB-GRANARY-001 and TOB-GRANARY-006.	**Weak**
Complexity Management	The code inherited from AAVE 2.0 has clear responsibilities and is organized to facilitate effective unit tests. The added code follows the same structure and conventions; however, the codebase is not yet complete. Problems with complexity have led to oversights and basic functionality issues, such as those highlighted in	**Satisfactory**

	TOB-GRANARY-003 and TOB-GRANARY-007. Areas of high complexity could use additional documentation or revisions to make them clearer.	
Decentralization	The codebase keeps the same design decision for decentralization as AAVE 2.0. For instance, users cannot deploy their own mini pools, similar to the lending pools. Additionally, there is a special privileged role that can change the specific configuration.	**Satisfactory**
Documentation	The codebase benefits from the extensive documentation available for AAVE 2.0; however, the specific modifications and new features need to be carefully documented to facilitate future audits and make users aware of the changes. In particular, more documentation is needed to cover in-depth features such as rehypothecation and mini pool interest rates.	**Moderate**
Low-Level Manipulation	Low-level bitwise manipulation is used across the codebase to handle configuration options. While this pattern is inherited from the AAVE 2.0 codebase, it should be used carefully to avoid introducing issues such as TOB-GRANARY-003. Additionally, assembly is heavily used in the Solady dependency to implement ERC-6909. The contract itself was out of scope, but there are concerns about the use of optimizations (TOB-GRANARY-008).	**Moderate**
Testing and Verification	The codebase contains several unit and integration tests. However, high-severity issues are still present, indicating that the test suite should be expanded. Additionally, there is an ongoing effort to use fuzz testing, which needs to be expanded to make sure that not only the new features are working as expected but also the original AAVE 2.0 invariants still hold.	**Moderate**
Transaction Ordering	Lending platforms in general are vulnerable to front-running issues; the codebase should undergo a more in-depth review to find these vulnerabilities.	**More Investigation Required**

Summary of Findings

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

ID	Title	Type	Severity
1	Lack of access controls when adding new reserves into ATokenERC6909	Access Control	High
2	Deposits can be blocked if the depositCapExponent value is too large	Data Validation	Medium
3	Incorrect bitwise operations used for the configuration of reserves	Undefined Behavior	Medium
4	Rebalancing triggers deposits or withdrawals in a vault without checking for limits	Data Validation	Medium
5	Missing return value in repay function	Undefined Behavior	Medium
6	Mini pool borrowing function has no access controls	Access Control	Medium
7	Mini pool debt bookkeeping is fragile	Undefined Behavior	High
8	Optimizations are enabled and could affect ERC-6909 bytecode	Undefined Behavior	Undetermined
9	The _rebalance function does not emit an event	Auditing and Logging	Informational
10	Incorrect use of onBehalfOf parameter in mint function	Undefined Behavior	Medium

Detailed Findings

1. Lack of access controls when adding new reserves into ATokenERC6909	
Severity: **High**	Difficulty: **Medium**
Type: Access Control	Finding ID: TOB-GRANARY-001
Target: `contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol`	

Description

Any user can add reserves into an `ATokenERC6909` contract. `ATokenERC6909` requires a specific initialization, where underlying assets are added one by one using the `initReserve` function.

```
  function initReserve(
    address underlyingAsset,
    string memory name,
    string memory symbol,
    uint8 decimals
  ) external returns (uint256 aTokenID, uint256 debtTokenID, bool isTranche){
    //require(msg.sender == address(_addressesProvider.getMiniPoolConfigurator()),
"Must Be Provider");
    (aTokenID,  debtTokenID, isTranche) = getIdForUnderlying(underlyingAsset);
    if(isTranche) {
      _totalTrancheTokens++;
      _isTranche[aTokenID] = true;
      _isTranche[debtTokenID] = true;
    } else {
      _totalUniqueTokens++;
    }
    _initializeATokenID(aTokenID, underlyingAsset, name, symbol, decimals);
    _initializeDebtTokenID(debtTokenID, underlyingAsset, name, symbol, decimals);
  }
```

Figure 1.1: The `initReserve` function from the `ATokenERC6909` contract

Only the mini pool configuration address should be able to run this function. However, the access control code for this function was commented out, so any user can call it.

Exploit Scenario

Eve calls `initReserve` to reset the reserves' configuration of an underlying asset, potentially misconfiguring the metadata of aTokens and debt tokens.

Recommendations

Short term, restore the commented line and add a proper unit test to make sure this issue will not occur again.

Long term, review access controls across each component, document which roles are supposed to call each component, and add extensive unit and fuzzing tests to make sure they work as expected.

2. Deposits can be blocked if the depositCapExponent value is too large

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRANARY-002
Target: `protocol/lendingpool/minipool/logic/MiniPoolValidationLogic.sol`, `protocol/libraries/logic/ValidationLogic.sol`	

Description

The calculation of the maximum deposit can overflow, breaking the deposit flow for all users.

Before a deposit is accepted, it must be validated. One important validation is the check for the maximum deposit value (cap). This value is computed using the `depositCapExponent` parameter:

```
  function validateDeposit(DataTypes.MiniPoolReserveData storage reserve, uint256
amount) external view {
    (bool isActive, bool isFrozen, ) = reserve.configuration.getFlags();
    uint256 depositCapExponent = reserve.configuration.getDepositCap();
    uint256 depositCap = depositCapExponent != 0 ? 10 ** (depositCapExponent) :
type(uint256).max;
    uint256 total = IAERC6909(reserve.aTokenAddress).totalSupply(reserve.aTokenID);
    uint256 newTotal = total + amount;
    require(amount != 0, Errors.VL_INVALID_AMOUNT);
    require(isActive, Errors.VL_NO_ACTIVE_RESERVE);
    require(!isFrozen, Errors.VL_RESERVE_FROZEN);
    require(newTotal < depositCap, Errors.VL_DEPOSIT_CAP_REACHED);
  }
```

Figure 2.1: The `validateDeposit` function (`protocol/lendingpool/minipool/logic/MiniPoolValidationLogic.sol`)

However, the calculation uses exponentiation, which can overflow a `uint256` variable if it is performed with a base larger than 10.

Exploit Scenario

Alice sets a `depositCapExponent` with a value larger than 69. Bob tries to deposit, but an overflow in the `validateDeposit` function produces a revert, blocking the operation. Alice must change the `depositCapExponent` to a smaller value to restore deposits.

Recommendations

Short term, limit the `depositCapExponent` parameter to a sane value or use base 2 to calculate the deposit cap.

Long term, manually review all the critical arithmetic computation for overflow and use fuzz testing to detect unexpected reverts.

3. Incorrect bitwise operations used for the configuration of reserves

Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-GRANARY-003
Target: `protocol/libraries/configuration/ReserveBorrowConfiguration.sol`, `protocol/libraries/configuration/ReserveConfiguration.sol`	

Description

A few instances of bitwise operations for setting and getting configuration values in the reserves are not implemented correctly.

Several configuration options are handled using bitwise operations to set and get their values. For instance, in the `ReserveBorrowConfiguration` library, the volatility tier value is handled using a mask with a single bit of zero (e.g., `0xD == 0b1101`):

```
library ReserveBorrowConfiguration {
  ...
  uint256 constant VOLATILITY_TIER_MASK =
    0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDFFFFFFFFFFFFFFFFFFFFFF; //
prettier-ignore
  ...
  uint256 constant MAX_VALID_VOLATILITY_TIER = 4;
  ...
  function setVolatilityTier(
        DataTypes.ReserveBorrowConfigurationMap memory self,
        uint256 volatilityTier
    ) internal pure {
        require (volatilityTier <= MAX_VALID_VOLATILITY_TIER,
Errors.RC_INVALID_VOLATILITY_TIER);
        self.data =
        (self.data & VOLATILITY_TIER_MASK) |
        (volatilityTier << VOLATILITY_TIER_START_BIT_POSITION);
  }
```

Figure 3.1: Part of the `ReserveBorrowConfiguration` library

However, the maximum value defined for that parameter is 4, which will not fit in a single bit.

Another example of these mistakes is located in the `ReserveConfiguration` library. This code allows setting an integer as an exponent to compute a maximum value for deposits:

```
library ReserveConfiguration {
  …
  uint256 constant DEPOSIT_CAP_MASK =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00FFFFFFFFFFFFFFFFFFFF; //
prettier-ignore

  /// @dev For the LTV, the start bit is 0 (up to 15), hence no bitshifting is
needed
  …
  uint256 constant MAX_VALID_DEPOSIT_CAP = 256;

 function setDepositCap(DataTypes.ReserveConfigurationMap memory self, uint256
depositCap)
    internal
    pure
  {
    require(depositCap <= MAX_VALID_DEPOSIT_CAP, Errors.RC_INVALID_DEPOSIT_CAP);

    self.data =
      (self.data & DEPOSIT_CAP_MASK) |
      (depositCap << DEPOSIT_CAP_START_BIT_POSITION);
  }
```

Figure 3.2: Part of the `ReserveConfiguration` library

However, the maximum value defined for that parameter is 256, which will not fit in a single byte.

Exploit Scenario

Alice sets a parameter for the volatility tier to 2, but since the wrong bit mask is used, the value stored is `0`. An external contract reading this information will read the wrong value.

Recommendations

Short term, correct the incorrect mask or the maximum values for the volatility tier and maximum deposit cap parameters.

Long term, add extensive unit and fuzzing tests for all the code that uses bitwise operations.

4. Rebalancing triggers deposits or withdrawals in a vault without checking for limits

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GRANARY-002
Target: `protocol/tokenization/AToken.sol`	

Description

The interaction with ERC-4626 vaults during liquidations can revert due to very large deposits or withdrawals. The ERC-4626 standard defines a set of operations and properties. One important function is `maxWithdraw(address)`, which is described below:

```
# maxWithdraw(address owner) public view returns (uint256)

Maximum amount of the underlying asset that can be withdrawn from the owner balance
in the Vault, through a withdraw call.

MUST return the maximum amount of assets that could be transferred from owner
through withdraw and not cause a revert, which MUST NOT be higher than the actual
maximum that would be accepted (it should underestimate if necessary).

MUST factor in both global and user-specific limits, like if withdrawals are
entirely disabled (even temporarily) it MUST return 0.
```

Figure 4.1: The `maxWithdraw` function definition from the ERC-4626 standard

aTokens will use an ERC-4626 vault during rebalancing to deposit or withdraw assets:

```
  /// @dev Rebalance so as to free _amountToWithdraw for a future transfer
  function _rebalance(uint256 _amountToWithdraw) internal {
    …
    // + means deposit, - means withdraw
    int256 netAssetMovement = int256(toDeposit) - int256(toWithdraw) -
int256(profit);
    if (netAssetMovement > 0) {
      vault.deposit(uint256(netAssetMovement), address(this));
    } else if (netAssetMovement < 0) {
      vault.withdraw(uint256(-netAssetMovement), address(this), address(this));
    }
```

Figure 4.2: Part of the `_rebalance` function that shows deposits and withdrawals to ERC-4626 vaults

However, the code does not consider the maximum deposits or withdrawals defined by the standard, causing the process to revert if such limits are reached.

Exploit Scenario

If rebalancing is executed during a liquidation and the ERC-4626 vault contains a fixed limit that is reached during that operation, the entire transaction will revert. The liquidation will fail, allowing users to keep their positions for longer.

Recommendations

Short term, consider disabling rehypothecation during critical operations such as liquidations.

Long term, use fuzz testing to detect unexpected reverts.

5. Missing return value in repay function

Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-GRANARY-006
Target: `protocol/lendingpool/minipool/MiniPool.sol`	

Description

The return value for the `repay` function is not set and will always be zero.

The `repay` function specifies a `uint256` return value in its function definition, which is expected to be the repaid amount.

```
267    function repay(
268     address asset,
269     bool reserveType,
270     uint256 amount,
271     address onBehalfOf
272    ) external override whenNotPaused returns (uint256) {
273     uint256 repayAmount = MiniPoolBorrowLogic.repay(
274       MiniPoolBorrowLogic.repayParams(
275         asset,
276         reserveType,
277         amount,
278         onBehalfOf,
279         _addressesProvider
280       ),
281       _reserves,
282       _usersConfig
283     );
284     _repayLendingPool(asset, reserveType, amount);
285
286
287    }
```

Figure 5.1: The `repay` function (`protocol/lendingpool/minipool/MiniPool.sol#L267–L287`)

However, the function's logic lacks an explicit return statement, causing the function to always return the default value of zero, which could be incorrect and misleading to other components or contracts interacting with the function.

This issue was discovered while analyzing the codebase with Slither.

Exploit Scenario
An external protocol integrates with the Cod3x Lend protocol and relies on the return value of the `repay` function to perform its calculations, which leads to incorrect calculations, resulting in financial loss.

Recommendations
Short term, explicitly set the return value for the `repay` function.

Long term, use the Slither static analyzer to catch common issues such as this one. Consider integrating a Slither scan into the project's CI pipeline, pre-commit hooks, or build scripts.

6. Mini pool borrowing function has no access controls

Severity: **Medium**	Difficulty: **Low**
Type: Access Control	Finding ID: TOB-GRANARY-006
Target: `protocol/lendingpool/LendingPool.sol`	

Description

The `miniPoolBorrow` function has no access controls and allows any user to borrow as a mini pool and thereby avoid liquidations.

The lending pool has a specific function, `miniPoolBorrow`, for mini pools to borrow assets:

```
function miniPoolBorrow(
  address asset,
  bool reserveType,
  uint256 amount,
  address miniPoolAddress,
  address aTokenAddress
) external override whenNotPaused {
  DataTypes.ReserveData storage reserve = _reserves[asset][reserveType];

  BorrowLogic.executeMiniPoolBorrow(
      BorrowLogic.ExecuteMiniPoolBorrowParams(
          asset,
          reserveType,
          amount,
          msg.sender,
          aTokenAddress,
          _addressesProvider,
          _reservesCount
      ),
      _reserves
  );
  _miniPoolsWithActiveLoans[miniPoolAddress] = true;
}
```

Figure 6.1: The `miniPoolBorrow` function from the `LendingPool` contract

However, this function has no access controls, which allows users to pretend they are a mini pool with some amount of debt. Among other things, this will allow users to arbitrarily increase the debt of the mini pools:

```
function liquidationCall(
  address collateralAsset,
  bool collateralAssetType,
  address debtAsset,
  bool debtAssetType,
  address user,
  uint256 debtToCover,
  bool receiveAToken
) external override whenNotPaused {
  if(_miniPoolsWithActiveLoans[user]){
    revert();
  }
 ...
```

Figure 6.2: Header of the `liquidationCall` function in the `LendingPool` contract

Exploit Scenario

To inappropriately raise the interest rate of an asset in the lending pool, Eve invokes the vulnerable `miniPoolBorrow` function to increase the debt amount of the mini pool.

Recommendations

Short term, add an access control check to prevent users from calling `miniPoolBorrow`.

Long term, review access controls across each component, document which roles are supposed to call each component, and add extensive unit and fuzzing tests to make sure they work as expected.

7. Mini pool debt bookkeeping is fragile

Severity: **High**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-GRANARY-007
Target: `protocol/lendingpool/minipool/MiniPool.sol`	

Description

The approach to keep track of the amount borrowed by each mini pool from the lending pool is not robust and can be manipulated by users.

The amount of assets that each mini pool has borrowed from the lending pool is individually tracked in each mini pool, which checks its own balance and updates the `_lendingPoolDebt` state variable.

```
function borrow(
  address asset,
  bool reserveType,
  uint256 amount,
  address onBehalfOf
) external override whenNotPaused {
  DataTypes.MiniPoolReserveData storage reserve = _reserves[asset];
  borrowVars memory vars;
  vars.aTokenAddress = reserve.aTokenAddress;
  require(vars.aTokenAddress != address(0), "Reserve not initialized");
  vars.availableLiquidity = IERC20(asset).balanceOf(vars.aTokenAddress);
  if(amount > vars.availableLiquidity &&
IAERC6909(reserve.aTokenAddress).isTranche(reserve.aTokenID)){
    address underlying = IAToken(asset).UNDERLYING_ASSET_ADDRESS();
    vars.LendingPool = _addressesProvider.getLendingPool();
    ILendingPool(vars.LendingPool).
      miniPoolBorrow(
        underlying,
        reserveType,
        amount.sub(vars.availableLiquidity),
        address(this),
        address(asset)
      );

      vars.amountRecieved = IERC20(underlying).balanceOf(address(this));
      _lendingPoolDebt[(underlying)] =
_lendingPoolDebt[(underlying)].add(vars.amountRecieved);
      …
```

Figure 7.1: Header of the `borrow` function in the `MiniPool` contract

Every time there is a `repay` call in the mini pool, the debt is decreased by calling the `repay` function from the lending pool:

```
function repay(
  address asset,
  bool reserveType,
  uint256 amount,
  address onBehalfOf
) external override whenNotPaused returns (uint256) {
  uint256 repayAmount = MiniPoolBorrowLogic.repay(
    MiniPoolBorrowLogic.repayParams(
      asset,
      reserveType,
      amount,
      onBehalfOf,
      _addressesProvider
    ),
    _reserves,
    _usersConfig
  );
  _repayLendingPool(asset, reserveType, amount);
}
```

Figure 7.2: The `repay` function from the `MiniPool` contract

However, this approach is not robust for a number of reasons:

- The implementation is incomplete: `_lendingPoolDebt` is never decreased or reset.
- Users can send tokens to the mini pool address before calling the `borrow` function (in the mini pool contract) to artificially increase the received amount.
- Users can also decrease a mini pool's real debt using the `repay` function on behalf of the mini pool contract. When the debt is reduced to zero, the `repay` function in the lending pool will always revert, leading to a DoS during liquidations.

Exploit Scenario

Eve forces a mini pool to borrow a small amount of debt. She immediately repays the debt on the mini pool's behalf. Any subsequent call to `repay` will fail until the amount of debt is positive again.

Recommendations

Short term, consider using the debt tracking functionality from the lending pool instead of implementing a new one in each mini pool. Make sure the debt amount is checked every time there is a `repay` call to avoid reverting.

Long term, use extensive fuzzing tests to detect an invalid state when a mini pool invariant is broken.

8. Optimizations are enabled and could affect ERC-6909 bytecode

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-GRANARY-006
Target: `contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol`	

Description

There have been several bugs with security implications related to optimizations. 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.

```
[profile.default]
src = 'contracts'
out = 'out'
libs = ['node_modules', 'lib']
test = 'tests/foundry'
cache_path = 'cache_forge'
optimizer = true
optimizer_runs = 200
solc_version = '0.8.23'
```

Figure 8.1: Foundry configuration file with optimizations enabled

High-severity security issues due to optimization bugs have occurred in the past. For example, 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 an incorrect bit shift was patched in Solidity 0.5.6. More recently, a bug due to the incorrect caching of Keccak-256 was reported.

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

There are likely latent bugs related to optimization and/or new bugs that will be introduced due to future optimizations. In particular, a concern we have is the use of Solidity assembly and inheritance in the `ATokenERC6909` contract.

```
function transferFrom(address from, address to, uint256 id, uint256 amount)
    public
    payable
    virtual
    returns (bool)
{
    _beforeTokenTransfer(from, to, id, amount);
    /// @solidity memory-safe-assembly
    assembly {
        // Compute the operator slot and load its value.
        mstore(0x34, _ERC6909_MASTER_SLOT_SEED)
        mstore(0x28, from)
        mstore(0x14, caller())
        // Check if the caller is an operator.
        if iszero(sload(keccak256(0x20, 0x34))) {
            // Compute the allowance slot and load its value.
            mstore(0x00, id)
            let allowanceSlot := keccak256(0x00, 0x54)
            let allowance_ := sload(allowanceSlot)
            …
```

Figure 8.2: Part of Solady's implementation of the ERC-6909 token standard

Exploit Scenario

A latent or future bug in Solidity compiler optimizations causes a security vulnerability in the interaction between inherited Solidity code and the Solady implementation in assembly.

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.

9. The _rebalance function does not emit an event

Severity: **Informational**	Difficulty: Low
Type: Auditing and Logging	Finding ID: TOB-GRANARY-009
Target: `protocol/tokenization/AToken.sol`	

Description

The `_rebalance` function of the `AToken` contract performs an important operation for rehypothecation without emitting an event.

```
  function _rebalance(uint256 _amountToWithdraw) internal {
    if (farmingPct == 0 && farmingBal == 0) {
      return;
    }
    …
    if (netAssetMovement > 0) {
      vault.deposit(uint256(netAssetMovement), address(this));
    } else if (netAssetMovement < 0) {
      vault.withdraw(uint256(-netAssetMovement), address(this), address(this));
    }
    // if we recorded profit, recalculate it for precision and distribute
    if (profit != 0) {
      // profit is ultimately (coll at hand) + (coll allocated to yield generator) -
(recorded total coll Amount in pool)
      profit = IERC20(_underlyingAsset).balanceOf(address(this)) + farmingBal -
underlyingAmount;
      if (profit != 0) {
        // distribute to profitHandler
        IERC20(_underlyingAsset).safeTransfer(profitHandler, profit);
      }
    }
```

Figure 9.1: The `_rebalance` function (`protocol/tokenization/AToken.sol#L439–L485`)

Exploit Scenario

An attacker breaks an internal invariant of the `_rebalance` function. Due to the lack of an associated event, the incident is not noticed until later, delaying the team's activation of its incident response plan.

Recommendations

Short term, have the `_rebalance` function emit an event when rehypothecation is activated.

Long term, review critical operations in the codebase and ensure that relevant information is consistently logged.

10. Incorrect use of onBehalfOf parameter in mint function

Severity: **Medium**	Difficulty: Low
Type: Undefined Behavior	Finding ID: TOB-GRANARY-010
Target: `protocol/tokenization/ERC6909/ATokenERC6909.sol`	

Description

The `mint` function defines an `onBehalfOf` parameter, which determines the recipient of minted tokens, but it is not always used as expected.

```
   316    function mint(address user, address onBehalfOf, uint256 id, uint256
amount, uint256 index) external returns (bool){
 317     require(msg.sender == address(POOL),
Errors.CT_CALLER_MUST_BE_LENDING_POOL);
 318     if (amount == 0) {
 319       return false;
 320     }
 321
 322     uint256 previousBalance;
 323
 324     if(id >= DebtTokenAddressableIDs) {
 325       if(onBehalfOf != user) {
 326         require(_borrowAllowances[id][onBehalfOf][user] >= amount,
Errors.BORROW_ALLOWANCE_NOT_ENOUGH);
 327         _decreaseBorrowAllowance(onBehalfOf, user, id, amount);
 328       }
 329       previousBalance = super.balanceOf(onBehalfOf, id);
 330       uint256 amountScaled = amount.rayDiv(index);
 331       require(amountScaled != 0, Errors.CT_INVALID_MINT_AMOUNT);
 332       _mint(user, id, amountScaled);
 333     } else {
 334       previousBalance = super.balanceOf(user, id);
 335       uint256 amountScaled = amount.rayDiv(index);
 336       require(amountScaled != 0, Errors.CT_INVALID_MINT_AMOUNT);
 337       _mint(user, id, amountScaled);
 338     }
 339
 340     return previousBalance == 0;
 341   }
```

Figure 10.1: The `mint` function (protocol/tokenization/ERC6909/ATokenERC6909.sol#L316–L341)

If the provided `user` address differs from the `onBehalfOf` address, the code does not correctly use the `onBehalfOf` address, as tokens are minted to the provided `user` address instead.

Exploit Scenario

Alice has permission from Bob to borrow from a mini pool on his account. Alice calls `borrow` using the `onBehalfOf` parameter with Bob's address, but she ends up with the debt tokens instead of Bob.

Recommendations

Short term, in cases where the provided user address differs from the `onBehalfOf` address, make sure the `mint` function uses the `onBehalfOf` address as the receiver of the minted tokens.

Long term, review the differences and similarities between the original AAVE 2.0 contracts and Cod3x Lend and make sure these are properly documented.

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
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
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
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

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. Fuzz Testing Suite Expansion Recommendations

This appendix provides recommendations for adding enhancements and features to the fuzz testing suite to increase its coverage of the lending pool and mini pool components' system states.

- Check the Echidna usage guide.
- Review and use the pre-built ERC-20 tests for the ERC-20 compatible contracts (e.g., ERC-6909).
- Review and convert the following Certora properties:
  - AAVE 2.0
  - AAVE 3.0
- Make sure to not introduce unnecessary bias when the state of the contract is modified.
  - Review the following code; it is unclear whether this code allows aToken transfers to the lending pool or mini pools:

```
    function randTransfer(LocalVars_UPTL memory vul, uint seedUser,
uint seedRecipient, uint seedAToken, uint seedAmt) public {
        randUpdatePriceAndTryLiquidate(vul);

        uint randUser = clampBetween(seedUser, 0 ,totalNbUsers);
        uint randRecipient = clampBetween(seedRecipient, 0
,totalNbUsers);
        uint randAToken = clampBetween(seedAToken, 0 ,totalNbTokens);

        User user = users[randUser];
        User recipient = users[randRecipient];
        AToken aToken = aTokens[randAToken];

        uint userBalanceATokenBefore = aToken.balanceOf(address(user));
        uint recipientBalanceATokenBefore =
aToken.balanceOf(address(recipient));

        uint randAmt = clampBetween(seedAmt, 0, userBalanceATokenBefore
* 2); // "300" : zero amt transfer possible

        (bool success, bytes memory data) = user.proxy(
            address(aToken),
            abi.encodeWithSelector(
```

```
                aToken.transfer.selector,
                address(recipient),
                randAmt
            )
        );
        …
```

Figure C.1: Part of the `randTransfer` function

- Allow the fuzzer to change market parameters, but bound the values if necessary.

- Use cheat code if necessary (e.g., to simulate multiple users using the `prank` feature), but be careful not overuse it since it can produce confusing code.

- Constantly review the coverage information gathered during each fuzzing run to identify code paths that have not been exercised (unreached code), and refine the fuzzing strategy based on the insights gained. This may involve adjusting fuzzing parameters, adding new test cases, or focusing on specific code paths.

D. 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 April 23 to April 25, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Cod3x Lend 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 10 issues described in this report, the Cod3x Lend team has resolved eight issues and has not resolved the remaining two issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Lack of access controls when adding new reserves into ATokenERC6909	Resolved
2	Deposits can be blocked if the depositCapExponent value is too large	Resolved
3	Incorrect bitwise operations used for the configuration of reserves	Resolved
4	Rebalancing triggers deposits or withdrawals in a vault without checking for limits	Unresolved
5	Missing return value in repay function	Resolved
6	Mini pool borrowing function has no access controls	Resolved
7	Mini pool debt bookkeeping is fragile	Resolved
8	Optimizations are enabled and could affect ERC-6909 bytecode	Unresolved
9	The _rebalance function does not emit an event	Resolved
10	Incorrect use of onBehalfOf parameter in mint function	Resolved

Detailed Fix Review Results

TOB-GRANARY-1: Lack of access controls when adding new reserves into ATokenERC6909
Resolved in commit `b208312`. The commented access control code-line was restored and the error message was updated.

TOB-GRANARY-2: Deposits can be blocked if the depositCapExponent value is too large
Resolved in commit `2af50d9`. The `MAX_VALID_DEPOSIT_CAP` value has been reduced from 256 to 50.

TOB-GRANARY-3: Incorrect bitwise operations used for the configuration of reserves
Resolved in commit `a726e40`. The `MAX_VALID_DEPOSIT_CAP` constant in the `ReserveConfiguration` file has been changed from 256 to 255, which allows the mask to accommodate the maximum value for the deposit cap. In commit `8336944`, The `VOLATILITY_TIER_MASK` constant in the `ReserveBorrowConfiguration` file has also been updated to accommodate 3 bits since the maximum value defined for the `volatilityTier` variable is 4.

TOB-GRANARY-4: Rebalancing triggers deposits or withdrawals in a vault without checking for limits
Unresolved. The issue has not been resolved. The Cod3x Lend team provided the following context for this finding's fix status:

> *The proposed solution of disabling rehypothecation during liquidations wouldn't help, though it's still a legitimate concern. Our solution is to ensure the vaults don't limit withdrawals more than they do naturally & to abide by best risk management practice. This involves ensuring there are enough reserves available to service liquidations under extremely volatile conditions.*

TOB-GRANARY-5: Missing return value in repay function
Resolved in commit `12d7e27`. The `repay` function has been updated to include a return statement that returns the calculated `repayAmount` value.

TOB-GRANARY-6: Mini pool borrowing function has no access controls
Resolved in commit `34876c7`. The fix added an access control on the `miniPoolBorrow` function to allow only the `minipool` address to call the function; it also defines some additional error codes.

TOB-GRANARY-7: Mini pool debt bookkeeping is fragile
Resolved in commit `b7a4022`. Debt tracking accounting between the minipool and the lending pool has been refactored. Rather than maintaining a debt record within the minipool contract, a new function has been introduced. This function calculates the minipool's current debt based on the value of its debt tokens during repayments.

TOB-GRANARY-8: Optimizations are enabled and could affect ERC-6909 bytecode
Unresolved. The issue has not been resolved. The Cod3x Lend team provided the following context for this finding's fix status:

> *Optimizations are enabled due to contract size issues, and while 'double optimizations' with 6909 contract may not be intended, the solady 6909 implementation is written by the EIPs author and I do believe has been audited as well. Since the end contract is upgradeable we also feel as though there are ways to mitigate if there ends up being a bug in the optimizer. We also feel like it's reasonable to when it is deployed to deploy 6909 separately than the rest of the protocol with optimizations disabled.*

TOB-GRANARY-9: The _rebalance function does not emit an event
Resolved in commit `d9e9130`. A new event is defined and is now emitted within the `_rebalance` function.

TOB-GRANARY-10: Incorrect use of onBehalfOf parameter in mint function
Resolved in commit `29e4b60` and `b338e62`. In these fixes, the recipient of the Atokens minted when the user address is different from the `onBehalfOf` address has been updated from the user address to the `onBehalfOf` address.

E. 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.