AstraSec

BakerDAO Security Audit Report

March 20, 2025

Contents

1 Introduction

1.1 About Bread Protocol

BakerDAO's Bread Protocol introduces a revolutionary DeFi primitive combining token economics, collateralized lending, leverage trading, and protocol-owned liquidity into a single cohesive financial ecosystem. The BREAD token features a mathematically-enforced upward price trajectory, backed by BERA tokens and sustainable economic incentives.

Additional Features Highlights:

- **FeeCollector/TWAP**: An externally callable function allowing users to claim any ERC20 token balance within the Bread contract (excluding BREAD) in exchange for a fixed BERA bounty. Inspired by the Uniswap Fee Collector, this mechanism enables Bread to accumulate additional backing through fees and revenues from liquid ERC20 assets beyond BERA.
- **Token Locker**: A lightweight ERC20 token locker integrated directly into the BREAD contract. It offers a basic token locking solution, particularly useful for newer tokens. A small percentage of each lock contributes to Bread's backing.
- **Bread Staking (iBREAD)**: Users can stake BREAD to mint iBREAD, an elastic-supply ERC20 token. Minting and burning operations are controlled by contracts authorized by the Bread contract owner. This feature remains inactive until explicitly enabled.
- **BreadGT (BGT Wrapper)**: A liquid wrapper for BGT on Berachain, allowing liquidity providers to authorize the Bread contract to claim BGT rewards on their behalf. The claimed BGT is stored in the breadTreasury, and breadGT receipt tokens are minted 1:1 in return.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/Gluten-Foundation/bakerdao-contracts
- CommitID: 90c425457b3e2464f4b7d5d01e8316883a27ee0f

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/Gluten-Foundation/bakerdao-contracts
- CommitID: c2674bd31dbc81fdecb783ac64364d5961c4cf73

1.3 Revision History

Version	Date	Description
v1.0	March 20, 2025	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Bread protocol. Throughout this audit, we identified a total of 7 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	–	–	–	–
High	–	–	–	–
Medium	2	–	–	2
Low	5	1	–	4
Informational	–	–	–	–
Total	7	1	–	6

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~	Incorrect toUser Amount Calculation in flashClosePosition()
~~M-2~~	Flawed Algorithm in leverage()
L-1	Potential Risks Associated with Centralization
~~L-2~~	Revised Calculation of Excess Collateral in borrowMore()
~~L-3~~	Improved Deduction of totalStakedBread in unstakeBread()
~~L-4~~	Incorrect Interest Calculation in borrow() Function
~~L-5~~	Incorrect Fee Rate Applied in flashClosePosition()

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

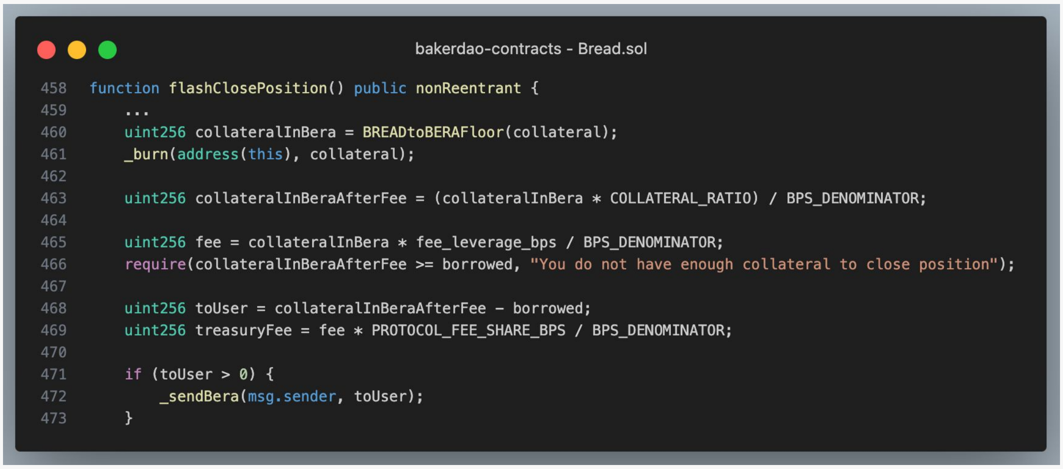
Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [M-1] Incorrect toUser Amount Calculation in flashClosePosition()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bread.sol	Algorithm	Medium	Medium	Addressed

The flashClosePosition() function is designed to liquidate a user's loan position by selling the collateral based on the current BREAD-to-BERA ratio, deducting a fee, reducing the borrowed amount, and returning the remaining BERA to the user. However, an issue has been identified in the calculation of the toUser amount. The variable collateralInBeraAfterFee, despite its name suggesting it represents the collateral value in BERA after the fee fee deduction, actually reflects the maximum borrowable value. This miscalculation can lead to incorrect BERA amounts being returned to users, potentially leading to financial loss or unintended protocol behavior, particularly under different fee configurations.


bakerdao-contracts - Bread.sol

```
function flashClosePosition() public nonReentrant {
    ...
    uint256 collateralInBera = BREADtoBERAFloor(collateral);
    _burn(address(this), collateral);

    uint256 collateralInBeraAfterFee = (collateralInBera * COLLATERAL_RATIO) / BPS_DENOMINATOR;

    uint256 fee = collateralInBera * fee_leverage_bps / BPS_DENOMINATOR;
    require(collateralInBeraAfterFee >= borrowed, "You do not have enough collateral to close position");

    uint256 toUser = collateralInBeraAfterFee - borrowed;
    uint256 treasuryFee = fee * PROTOCOL_FEE_SHARE_BPS / BPS_DENOMINATOR;

    if (toUser > 0) {
        _sendBera(msg.sender, toUser);
    }
```

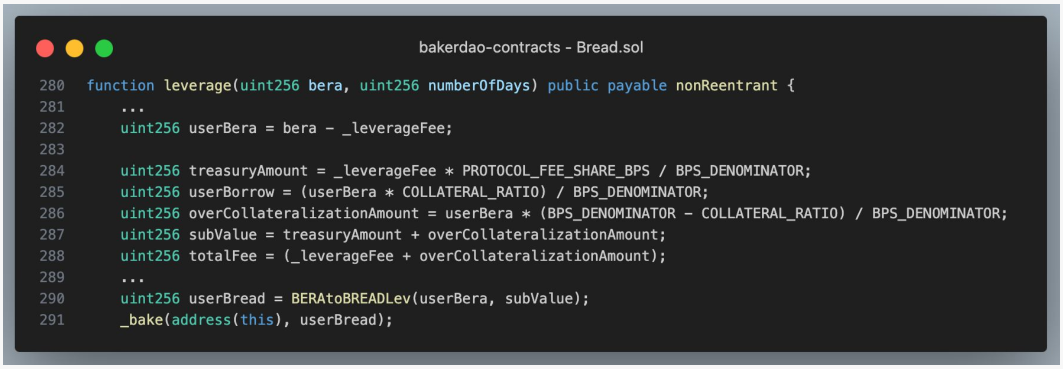
Remediation To address this issue, we recommend modifying the calculation of the collateralInBeraAfterFee variable to accurately reflect the collateral value in BERA after the fee.

3.3.2 [M-2] Flawed Algorithm in leverage()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bread.sol	Algorithm	Medium	Medium	Addressed

The leverage() function enables users to open a leveraged position by paying fees upfront, with the protocol recording a corresponding loan on behalf of the user. However, three significant issues have been identified in the implementation:

- Incorrect userBera Calculation: The userBera amount, intended to represent the BERA available for calculating the borrow amount after deducting a mintFee from the input bera amount, is incorrectly reduced by both the mintFee and the interest. This overcharging reduces the borrowable amount and results in an inaccurate loan calculation.
- Miscalculated subValue: The subValue, which denotes the BERA amount a user must pay for leverage, is miscalculated. It should match the totalFee, but the existing logic flaws result in an erroneous computation, subsequently affecting the accuracy of the userBread amount.
- Inappropriate Rounding in BERAtoBREADLev(): When calculating the BREAD amount using the BERAtoBREADLev() function, it uses a rounding-up value, which favors the user. Since the BREAD amount serves as the collateral for the loan, it is expected to use a default rounding-down value to favor the protocol, ensuring a conservative approach to collateral valuation.


bakerdao-contracts - Bread.sol

```
function leverage(uint256 bera, uint256 numberOfDays) public payable nonReentrant {
    ...
    uint256 userBera = bera - _leverageFee;

    uint256 treasuryAmount = _leverageFee * PROTOCOL_FEE_SHARE_BPS / BPS_DENOMINATOR;
    uint256 userBorrow = (userBera * COLLATERAL_RATIO) / BPS_DENOMINATOR;
    uint256 overCollateralizationAmount = userBera * (BPS_DENOMINATOR - COLLATERAL_RATIO) / BPS_DENOMINATOR;
    uint256 subValue = treasuryAmount + overCollateralizationAmount;
    uint256 totalFee = (_leverageFee + overCollateralizationAmount);
    ...
    uint256 userBread = BERAtoBREADLev(userBera, subValue);
    _bake(address(this), userBread);
```

Remediation To ensure the leverage() function aligns with the intended leverage design, we recommend addressing the above mentioned issues.

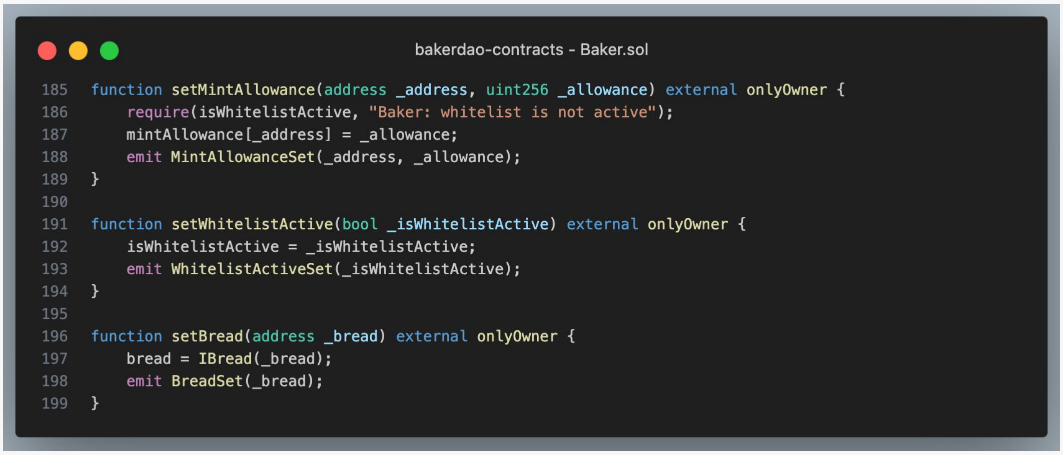
3.3.3 [L-1] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Low	Acknowledged

The Bread protocol exhibits centralization risks due to its heavy reliance on a privileged owner account for configuring and managing key contract parameters. The owner is granted extensive control over protocol-wide operations, which introduces potential vulnerabilities. Specifically, the owner can perform the following actions:

- Set different fees, treasury address, baker address, max supply for the Bread contract.
- Set mint allowance, deadline, bread address for the Baker contract.
- Other privileged operations.

This concentration of power in a single entity increases the risk of unauthorized or malicious modifications, potentially compromising the protocol's integrity and fairness.


bakerdao-contracts - Baker.sol

```
function setMintAllowance(address _address, uint256 _allowance) external onlyOwner {
    require(isWhitelistActive, "Baker: whitelist is not active");
    mintAllowance[_address] = _allowance;
    emit MintAllowanceSet(_address, _allowance);
}

function setWhitelistActive(bool _isWhitelistActive) external onlyOwner {
    isWhitelistActive = _isWhitelistActive;
    emit WhitelistActiveSet(_isWhitelistActive);
}

function setBread(address _bread) external onlyOwner {
    bread = IBread(_bread);
    emit BreadSet(_bread);
}
```

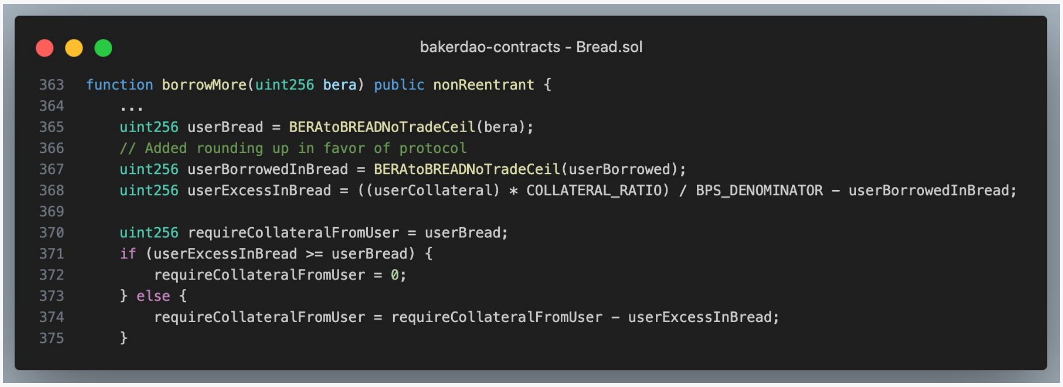
Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Team Response This issue has been acknowledged by the team and is an intentional control mechanism as per the protocol design.

3.3.4 [L-2] Revised Calculation of Excess Collateral in borrowMore()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bread.sol	Algorithm	Low	Low	Addressed

In the Bread protocol, the borrowMore() function allows users to borrow more BERA based on their current loan position. However, there's a miscalculation in determining userExcessInBread, intended to represent the surplus collateral available to the user in BREAD. The current implementation mistakenly calculates userExcessInBread as the excess borrow amount in BREAD, rather than the excess collateral amount in BREAD. As a result, this miscalculation can lead to an inflated requireCollateralFromUser value, causing users to provide more BREAD as collateral than necessary when invoking the borrowMore() function.



bakerdao-contracts - Bread.sol

```
function borrowMore(uint256 bera) public nonReentrant {
    ...
    uint256 userBread = BERAtoBREADNoTradeCeil(bera);
    // Added rounding up in favor of protocol
    uint256 userBorrowedInBread = BERAtoBREADNoTradeCeil(userBorrowed);
    uint256 userExcessInBread = ((userCollateral) * COLLATERAL_RATIO) / BPS_DENOMINATOR - userBorrowedInBread;

    uint256 requireCollateralFromUser = userBread;
    if (userExcessInBread >= userBread) {
        requireCollateralFromUser = 0;
    } else {
        requireCollateralFromUser = requireCollateralFromUser - userExcessInBread;
    }
```

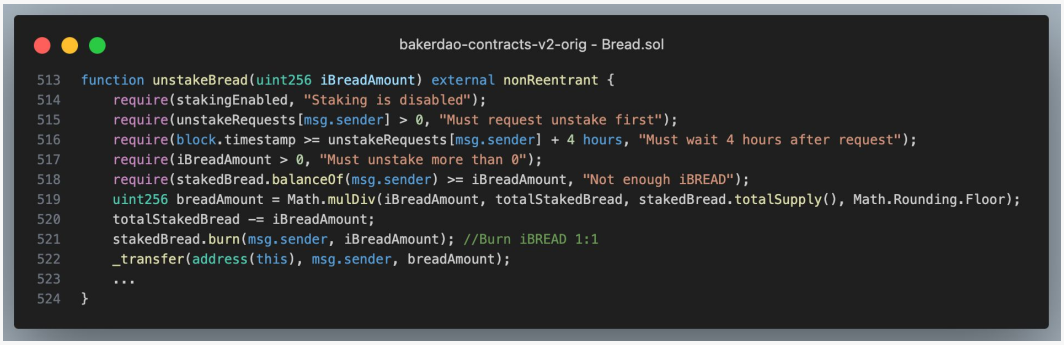
Remediation To align the calculation with the intended design of representing excess collateral, modify the userExcessInBread calculation as follows:

*userExcessInBread = userCollateral - (userBorrowedInBread * BPS_DENOMINATOR / COLLATERAL_RATIO);*

3.3.5 [L-3] Improved Deduction of totalStakedBread in unstakeBread()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bread.sol	Algorithm	Low	Low	Addressed

The Bread contract enables users to stake BREAD tokens to mint iBREAD tokens and burn iBREAD to withdraw BREAD tokens. The totalStakedBread state variable is designed to track the total amount of BREAD staked within the contract. However, in the unstakeBread() function, the totalStakedBread is mistakenly reduced by the iBreadAmount (the amount of iBREAD being burned) instead of the breadAmount (the actual amount of BREAD being withdrawn). This is problematic because iBreadAmount and breadAmount may may differ due to fluctuations in the BREAD/iBREAD ratio, which is influenced by the contract's internal logic. As a result, totalStakedBread becomes inaccurate over time, potentially disrupting accounting accuracy and fairness in future staking and unstaking operations.


bakerdao-contracts-v2-orig - Bread.sol

```solidity
function unstakeBread(uint256 iBreadAmount) external nonReentrant {
    require(stakingEnabled, "Staking is disabled");
    require(unstakeRequests[msg.sender] > 0, "Must request unstake first");
    require(block.timestamp >= unstakeRequests[msg.sender] + 4 hours, "Must wait 4 hours after request");
    require(iBreadAmount > 0, "Must unstake more than 0");
    require(stakedBread.balanceOf(msg.sender) >= iBreadAmount, "Not enough iBREAD");
    uint256 breadAmount = Math.mulDiv(iBreadAmount, totalStakedBread, stakedBread.totalSupply(), Math.Rounding.Floor);
    totalStakedBread -= iBreadAmount;
    stakedBread.burn(msg.sender, iBreadAmount); //Burn iBREAD 1:1
    _transfer(address(this), msg.sender, breadAmount);
    ...
}
```

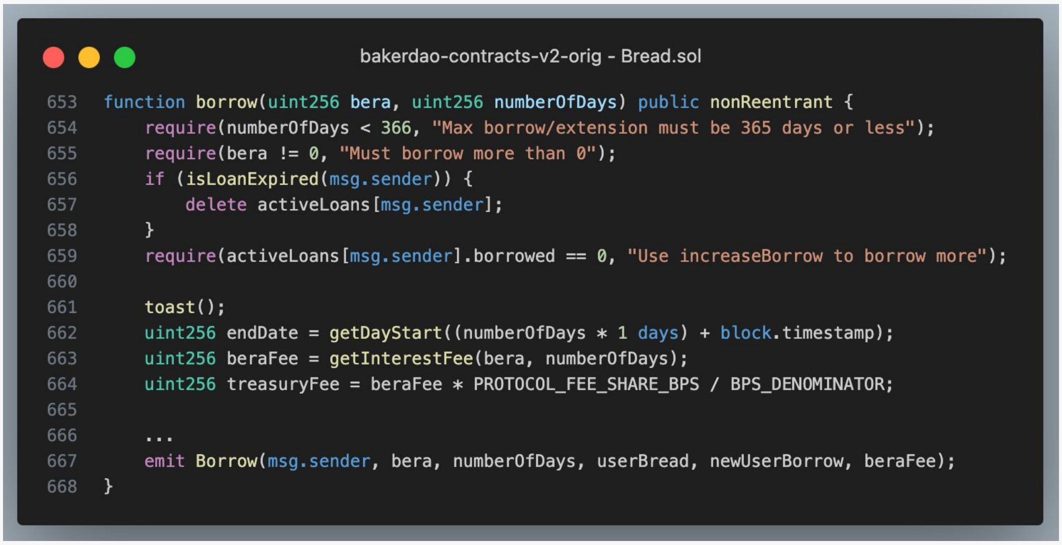
Remediation Revisit the unstakeBread() function to deduct the correct breadAmount from totalStakedBread, instead of the iBreadAmount.

3.3.6 [L-4] Incorrect Interest Calculation in borrow() Function

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bread.sol	Business Logic	Low	Low	Addressed

In the Bread contract, the borrow() function allows users to supply BREAD as collateral to borrow BERA, with interest charged upfront based on the borrowed amount and loan duration. However, during the audit, an issue was identified in the interest calculation logic. The interest fee (beraFee) is calculated using the underlying BERA value (bera) instead of the actual borrowed BERA amount (newUserBorrow). This miscalculation inflates the interest charged, causing borrowers to pay more than they should.

Moreover, this issue is not limited to the borrow() function. Similar incorrect interest calculations are present in related functions, including borrowMore(), extendLoan(), and leverageFee().



```
function borrow(uint256 bera, uint256 numberOfDays) public nonReentrant {
    require(numberOfDays < 366, "Max borrow/extension must be 365 days or less");
    require(bera != 0, "Must borrow more than 0");
    if (isLoanExpired(msg.sender)) {
        delete activeLoans[msg.sender];
    }
    require(activeLoans[msg.sender].borrowed == 0, "Use increaseBorrow to borrow more");

    toast();
    uint256 endDate = getDayStart((numberOfDays * 1 days) + block.timestamp);
    uint256 beraFee = getInterestFee(bera, numberOfDays);
    uint256 treasuryFee = beraFee * PROTOCOL_FEE_SHARE_BPS / BPS_DENOMINATOR;

    ...
    emit Borrow(msg.sender, bera, numberOfDays, userBread, newUserBorrow, beraFee);
}
```

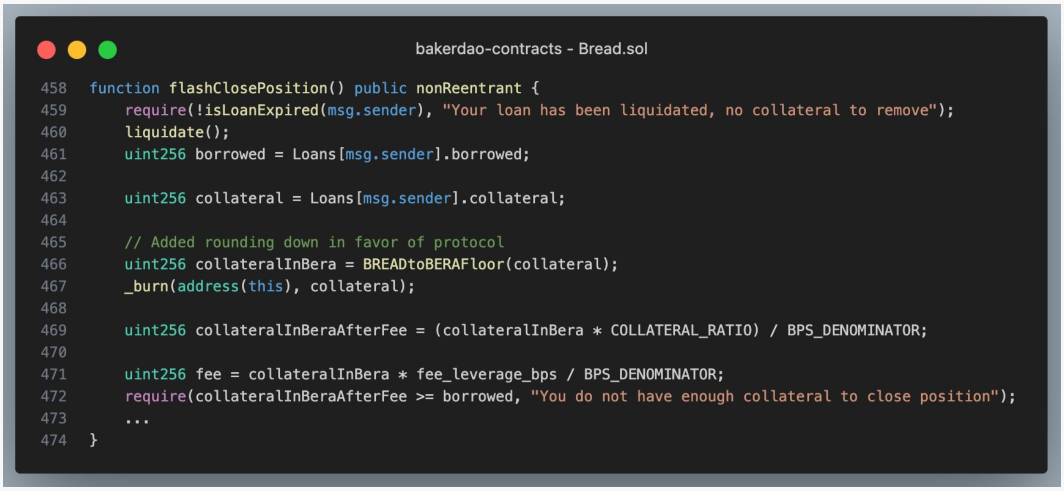
Remediation Revisit and update the borrow() function, along with related functions such as borrowMore(), extendLoan(), and leverageFee(), to ensure the interest fee is calculated based on the actual borrowed amount instead of the underlying BERA value.

3.3.7 [L-5] Incorrect Fee Rate Applied in flashClosePosition()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Bread.sol	Business Logic	Low	Low	Addressed

In the Bread contract, the flashClosePosition() function allows users to flash close a loan by selling their collateral (BREAD) to repay the borrowed amount (BERA), with any excess BERA returned to the user. As part of this process, a fee is charged to the user. However, the current implementation incorrectly applies the fee_leverage_bps rate, which is designed for leveraged borrowing, instead of the sell_fee_bps rate, which is intended for BREAD selling operations.

The fee_leverage_bps rate is typically lower than the sell_fee_bps rate, meaning the contract charges a smaller fee than expected for the operation. This discrepancy may result in reduced protocol revenue and may disrupt the intended economic incentives of the contract.


bakerdao-contracts - Bread.sol

```
function flashClosePosition() public nonReentrant {
    require(!isLoanExpired(msg.sender), "Your loan has been liquidated, no collateral to remove");
    liquidate();
    uint256 borrowed = Loans[msg.sender].borrowed;

    uint256 collateral = Loans[msg.sender].collateral;

    // Added rounding down in favor of protocol
    uint256 collateralInBera = BREADtoBERAFloor(collateral);
    _burn(address(this), collateral);

    uint256 collateralInBeraAfterFee = (collateralInBera * COLLATERAL_RATIO) / BPS_DENOMINATOR;

    uint256 fee = collateralInBera * fee_leverage_bps / BPS_DENOMINATOR;
    require(collateralInBeraAfterFee >= borrowed, "You do not have enough collateral to close position");
    ...
}
```

Remediation Revisit and update the flashClosePosition() function to calculate the fee using sell_fee_bps instead of fee_leverage_bps, as the operation involves selling BREAD collateral to repay the loan. This correction will align the fee calculation with the intended economic model and maintain the protocol's expected revenue structure.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI