FEATURE AUDIT - AAVE-V3 ‘BORROW()‘

Valya Zaitseva

January 30, 2026

Feature scope

1. The feature under the audit: `borrow()`.
2. Economic purpose: it allows to borrow assets from a protocol for users themself or on behalf of users provided that they were explicitly delegated. The protocol allows to borrow as long as it stays solvent.
3. State modified:
    - protocol liquidity,
    - total debt amount,
    - user debt balances and borrowing configuration

Invariants (grouped)

1. Pre-state (before borrow)

- PROTOCOL-LEVEL SAFETY: reserve should not be paused or frozen +
- PROTOCOL-LEVEL SAFETY: borrow should be allowed for reserve +
- PROTOCOL-LEVEL SAFETY: borrow should be allowed for an asset +
- ORACLE / PRICE: oracle should be set and price should not be zero +
- USER ACCOUNTING: user must have a borrowing power > borrow amount +
- ECONOMIC SAFETY: protocol should be solvent if it borrows some amount +
- PROTOCOL-LEVEL SAFETY: eMode category must be consistent with user selection +

2. Mid-process (during borrow)

- ORDERING: validate before debt mint before transferUnderlyingTo +
- AUTHORIZATION: user should be eligible to borrow +

3. Post-state (after borrow)

- PROTOCOL SOLVENCY: borrowed amount per debt token type == debt minted +
- PROTOCOL SOLVENCY: debt increased == debt minted +
- PROTOCOL SOLVENCY: liquidity decrease == debt minted +
- ACCOUNTING: transferred amount == debt minted +
- ACCOUNTING: Health factor >= liquidation threshold +

Failure Surface

If this feature fails, how does the protocol lose money?

- bad debt / insolvency,
- unauthorized debt creation,
- health factor below safe threshold, enabling liquidations,
- oracle-based over-borrowing.

Cross-Feature Invariant Consistency

`repay()` should reverse all borrow-induced state changes: decrease debt tokens and restore liquidity. `repay()` feature is not yet audited, but the invariant should be verified in a future audit cycle.

Call map with enforcement/assumption map - Execution Flow

```
borrow()
    -> executeBorrow()                                  <- * ENFORCED: ORDERING validate before debt mint before transferUnderlyingTo
        -> reserve.updateState()
        -> userConfig.getIsolationModeState()
        -> ValidationLogic.validateBorrow()             <- * ENFORCED: reserve should not be paused or frozen
                                                           * ENFORCED: borrow should be allowed for reserve
                                                           * ENFORCED: borrow should be allowed for an asset
                                                           * ENFORCED: protocol should be solvent if it borrows some amount
                                                           * ENFORCED: eMode is valid
                                                           * ENFORCED: HF >= liquidation threshold
                                                           * ENFORCED: user should have a borrowing power > borrow amount
                                                           * ENFORECED: oracle should be set and price should not be zero
                                                           * ASSUMED: user should be eligible to borrow - DELEGATION IS CHECKED IN DEBT TOKEN MINT
            -> GenericLogic.calculateUserAccountData    <- * ENFORCED: oracle should be set and price should not be zero
        -> IStableDebtToken.mint()                      <- * ENFORCED: user should be eligible to borrow
                                                           * ENFORCED: borrowed amount == debt minted
                                                           * ENFORCED: debt increased == debt minted
        -> IVariableDebtToken.mint()                    <- * ENFORCED: user should be eligible to borrow
                                                           * ENFORCED: borrowed amount == debt minted
                                                           * ENFORCED: debt increased == debt minted
        -> userConfig.setBorrowing()
        -> reserve.updateInterestRates()
            -> IReserveInterestRateStrategy.calculateInterestRates <- * ENFORCED: liquidity decrease == debt minted
        -> transferUnderlyingTo()                       <- * ENFORCED: transfered amount == debt minted
```

Sequence Flow

```
borrow()
 -> executeBorrow()
    -> reserve.updateState()                            -> accrues interest, updates indeces (storage write)
    -> userConfig.getIsolationModeState()               -> reads isolation mode state
    -> ACCOUNTING BOUNDARIES: ValidationLogic.validateBorrow() -> pre-state invariants enforced
    -> GenericLogic.calculateUserAccountData            -> computes HF, collateral, debt in base currency
    -> IRREVERSIBLE/EXTERNAL: IStableDebtToken.mint()   -> stable debt path, accrues past interest and mints new principal
    -> IRREVERSIBLE/EXTERNAL: IVariableDebtToken.mint() -> variable debt path, mints scaled principal using index
    -> userConfig.setBorrowing()                        -> updates user config if first borrow
    -> reserve.updateInterestRates()                    -> updates protocol rates and utilizaiton
    -> EXTERNAL: IAToken.transferUnderlyingTo()         -> transfers real underlying assets to borrower
```

Risk Hotspots - Threat Modeling

1. Oracle dependency - price must be fresh; stale prices can allow over-borrowing if sentinel not set.
2. Rounding issues - calculation involve base currency conversions, scaled balances, and LTV percentages.
3. Reentrancy - ordering ensures a reentrancy safe flow: validation -> debt mint -> transfer underlying.
4. Unauthorized borrow - delegation logic is checked in debt token mint functions.
5. Protocol insolvency - borrow caps, isolation mode ceilings, HF checkes enforced.
6. Edge cases - zero amount borrow is rejected.

5-lenses questions

1. Authorization Lens:
    1. Can debt be created without delegation? - NO
2. Accounting Lens:
    1. Can debt be minted without increasing total debt? - NO
    2. Can liquidity be modified incorrecly when transfer underlying? - NO
    3. Can transferred amount differ from minted debt? - NO
3. Economic Safety Lens:
    1. Can HF fall below threshold? - NO
    2. Can borrow cap be bypassed? - NO
    3. Can isolationModeDebtCeiling be bypassed? - NO
4. Oracle & External Dependency Lens
    1. Can a stale price allow over-borrowing? - YES - vars.collateralNeededInBaseCurrency depends on an asset price.
    2. Is there a sentinel/pause mechanism? - PARTIAL
    3. Is oracle checked before state changes? - YES
5. Temporal / Ordering Lens:
    1. Can debt mint happen before validation? - NO
    2. Can state updates be skipped? - NO
    3. Can reentrancy reoder logic? - No

Stable vs Variable Debt Notes

Debt Type	Mint Behavior	Interest Accounting
Stable Debt Token	mints principal + accrued interest	interest realized on mint, user balance grows immediately
Variable Debt Token	mints scaled principal using curent borrow index	interest implicit via index, events show ‘real’ debt, storage tracks scaled amount

Final verdict

- All major invariants are enforced in the borrow flow.
- No obvious invariant violations found.