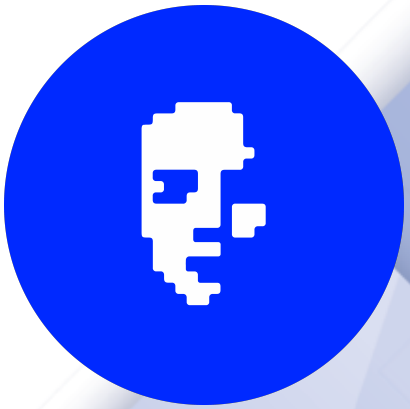
Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR



3Jane Morpho Blue

Veridise Inc.

Aug. 29, 2025

► **Prepared For:**

3Jane
https://www.3jane.xyz

► **Prepared By:**

Aayushman Thapa Magar
Ben Mariano
Ben Sepanski
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Aug. 29, 2025	V2
Aug. 25, 2025	V1
Aug. 22, 2025	Initial Draft

Contents

1 Executive Summary

From Aug. 7, 2025 to Aug. 18, 2025, 3Jane engaged Veridise to conduct a security assessment of the 3Jane Morpho Blue protocol. The security assessment covered the smart contracts implementing their under-collateralized lending platform 3Jane Morpho Blue. Veridise conducted the assessment over 20 person-days, with 2 security analysts reviewing the project over 10 days on commit `2d2825fe` in the 3Jane Morpho Blue repository and commit `10f8d1d8` in the 3Jane Morpho Blue USD3 repository. The review strategy involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as thorough code review.

Project Summary. The security assessment reviewed the on-chain components of the 3Jane Morpho Blue protocol that implement its under-collateralized lending system. In 3Jane Morpho Blue, three entities are involved: borrowers, lenders, and the protocol administrator. Lenders deposit USDC as liquidity in exchange for interest, while borrowers receive credit lines set by the protocol's administrator. Unlike traditional over-collateralized lending, borrowing limits are based on a user's off-chain reputation and financial records.

This design depends on two off-chain processes that were out of scope for the review:

- zkTLS proofs are used to privately verify bank statements and credit scores.
- Debt collection is handled by U.S.-based agencies if a borrower defaults.

On-chain, the protocol supports four key workflows: borrowing, repayment, supplying liquidity, and withdrawing liquidity.

Funds are managed within a single active `Market` via the `MorphoCredit` contract.

Users interact with the system through two contracts:

- `USD3` – for liquidity providers, who deposit USDC and receive `USD3` or `sUSD3` tokens in return.
- `Helper` – for borrowers, who take out and repay loans based on the rules of the associated `CreditLine`.

Interest accrues continuously based on the borrower's credit rating and any penalties. Borrowers must meet monthly minimum payments, with late or missed payments moving loans through delinquency and default stages. During default, the `MarkdownManager` gradually reduces the value of supply assets to ease liquidation and prevent sharp market shocks.

More details can be found in the 3Jane Morpho Blue whitepaper*, with further discussion provided in Section 4.

* `https://www.3jane.xyz/pdf/whitepaper.pdf`

Code Assessment. The 3Jane Morpho Blue developers provided the source code of the 3Jane Morpho Blue contracts for the code review. Most of the source code is original code written by the 3Jane Morpho Blue developers. Two modules build on and extend the logic of third-party codebases. The `MorphoCredit` contract modifies and extends the `Morpho` contract [†]. The `USD3` contract extends the Yearn Tokenized Strategy [‡], which handles the basic ERC4626 tokenized vault logic.

The codebases contain detailed inline documentation on contracts, functions, and variables indicating their intended usage. The 3Jane Morpho Blue white-paper provides a clear, high-level description of the system. The source code contained a test suite, which the Veridise security analysts noted was high quality. The developers clearly have automated tools in place for fuzzing, linting, and static analysis. The tests are high-coverage, with all workflows tested and several tests checking core protocol invariants. The computed interest rates and accruals are heavily tested to ensure computations are correct. The suite also includes negative tests checking that expected checks are enforced, such as access control and bounds on parameters.

Summary of Issues Detected. The security assessment uncovered 10 issues, 3 of which are assessed to be of high or critical severity. For example, issue V-3JNE-VUL-001 allows attackers to create fake markets that appear funded, enabling them to drain the protocol's waUSDC wallet. Issue V-3JNE-VUL-002 can enable a malicious user to prevent legitimate users from accessing their funds by making tiny deposits on their behalf and continuously resetting withdrawal lock timers.

The Veridise analysts also identified 2 medium-severity issues. For instance, issue V-3JNE-VUL-005 involves a storage slot mismatch in Morpho's helper libraries, which causes misaligned variable mappings that could break contract functionality for integrators. In issue V-3JNE-VUL-004, a flaw was found that, in case of an empty market, allows a malicious user to mint borrow shares without receiving any underlying assets, potentially driving the protocol into a denial-of-service state.

Additionally, there are 5 warnings.

The 3Jane Morpho Blue developers have provided fixes for many of the identified issues as they were reported.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the 3Jane Morpho Blue.

Splitting roles. Some roles have a wide variety of permissions. For example, the `ProtocolConfig.owner` is responsible for pausing the protocol, but also may change key protocol configuration settings. This role should be given to a smart contract so that the individual privileges may be split up. This will allow deployers to use different keys and delay settings for actions with varying levels of sensitivity. Section 4 contains a longer discussion.

Least privileges approach. One of the most severe issues comes from lack of access control on market creation. In future versions, any operations which are not intended for use in the regular

[†] https://github.com/morpho-org/morpho-blue

[‡] https://github.com/yearn/tokenized-strategyhttps://etherscan.io/address/0xD377919FA87120584B21279a491F82D5265A139c#code

user workflow should be removed or access controlled. This will prevent attackers from taking advantage of any unexpected misuse of these interfaces.

Improve Code Maintainability. The audit identified several maintainability improvements that could be made to the development practices of the system. Issues such as misaligned storage slots, unchecked return values, unused code, missing getters, and setter functions without validation or event emissions increase the likelihood of misconfigurations and make the protocol harder to maintain over time. While these findings are less immediately exploitable, they undermine transparency and operational reliability. Enforcing stricter validation, ensuring consistent event logging, and removing unused or redundant code will simplify integrations, and improve the protocol's long-term stability.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
3Jane Morpho Blue	`2d2825fe`	Solidity	Ethereum
3Jane Morpho Blue USD3	`10f8d1d8`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Aug. 7–Aug. 18, 2025	Manual & Tools	2	20 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	1	1	1
High-Severity Issues	2	2	2
Medium-Severity Issues	2	2	2
Low-Severity Issues	0	0	0
Warning-Severity Issues	5	5	1
Informational-Severity Issues	0	0	0
TOTAL	10	10	6

Table 2.4: Category Breakdown.

Name	Number
Maintainability	4
Data Validation	3
Logic Error	2
Access Control	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of 3Jane Morpho Blue's source code. During the assessment, the security analysts aimed to answer questions such as:

- Are any common Solidity security vulnerabilities such as reentrancy, access control, price manipulation, front-running, or locked funds present in the codebase?
- Can borrowers obtain credit lines without being approved by the `CreditLine` contract?
- Is the vault vulnerable to first depositor attacks?
- Can the value of debt be decreased by a user without repayment?
- Can the value of liquidity shares be manipulated?
- Can attackers increase another user's debt balance?
- Are debt, interest, and fees properly accounted?
- Can borrowers take out more loans than they are approved for?
- Can borrows circumvent the minimum required payments?
- Are transfer limitations on `USD3` and `sUSD3` properly implemented?
- Can malicious users leverage non-3Jane Morpho Blue `Markets` to attack the protocol?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a combination of human experts and automated program analysis & testing tools. In particular, the security assessment was conducted with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, security analysts leveraged Veridise's custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* Security analysts leveraged fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, the desired behavior of the protocol was formulated as [V] specifications and then tested using Veridise's fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this security assessment is limited to the following locations, which contain the smart contract implementation of the 3Jane Morpho Blue:

- The `src/` directory in 3Jane Morpho Blue Morpho Blue.
- The `src/` directory in 3Jane Morpho Blue USD3.

Methodology. Veridise security analysts reviewed the reports of previous audits for 3Jane Morpho Blue, inspected the provided tests, and read the 3Jane Morpho Blue documentation. They then began a review of the code assisted by both static analyzers and automated testing.

Before the security assessment, the Veridise security analysts met with the 3Jane Morpho Blue developers to ask questions about the code. During the review, the Veridise team discussed further quesetions asynchronously with the 3Jane developers.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Trust Model

4.1 Operational Assumptions.

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling security for 3Jane Morpho Blue.

- The credit system only approves users for which legal enforcement of the debt will be feasible, and approves credit lines which they may reasonably repay.
- Debt, once defaulted, is auctioned off to collections agencies to recoup losses.
- Any markdown manager used implements logic which decays debt from its initial value to zero.
- The administrators will set minimum repayment amounts based on the provided formulas*.

4.2 Privileged Roles.

Roles. This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensitivity. *Highly-privileged* roles may have a critical impact on the protocol if compromised, while *limited-authority* roles have a negative, but manageable impact if compromised. Time-sensitive *emergency* roles may be required to perform actions quickly based on real-time monitoring, while *non-emergency* roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assume that the role operators perform their responsibilities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- Highly-privileged, emergency roles:
  - The `ProtocolConfig.owner` is responsible for pausing the protocol in case of an emergency. Additionally, the owner is responsible for setting bounds on a variety of parameters including the length of grace/delinquency periods, global bounds on loan sizes, tranche ratios in case of default leading to supplier losses, and interest rate configurations. All configurations except for the paused status are not required for emergencies, but are held by the same role.
- Highly-privileged, non-emergency roles:
  - The `CreditLine.owner` is responsible for setting the contracts used to approve credit lines. Additionally, they may post minimum repayments after each payment cycle ends. These repayments are computed as a fraction of a user's loan, with each fraction computed off-chain by the `CreditLine.owner`. Finally, the `CreditLine`'s owner may settle debt by covering users' debt from an insurance fund.

* `https://www.3jane.xyz/pdf/whitepaper.pdf`

- Additionally, all the core contracts, including `MorphoCredit`, `CreditLine`, and `USD3`, are upgradeable. Only the owner may upgrade the contract, changing its logic arbitrarily.

Operational Recommendations. Highly-privileged, non-emergency operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

The 3Jane team should consider setting a smart contract as the owner for each core contract. This will allow enforcing different delays when changing key protocol parameters or performing upgrades, and ensure keys used for more frequent actions like pausing do not also have the authority to perform configuration changes which may damage the entire protocol if compromised.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-3JNE-VUL-001	Funds draining via malicious market creation	Critical	Fixed
V-3JNE-VUL-002	Griefing attack preventing withdrawals . . .	High	Fixed
V-3JNE-VUL-003	USD3/sUSD3 lock period bypass via . . .	High	Fixed
V-3JNE-VUL-004	Denial-of-service due to borrowed shares . . .	Medium	Fixed
V-3JNE-VUL-005	Storage slots indexes mismatch	Medium	Fixed
V-3JNE-VUL-006	Callback onMorphoSupply is called before . . .	Warning	Acknowledged
V-3JNE-VUL-007	Missing event emissions for important . . .	Warning	Acknowledged
V-3JNE-VUL-008	Missing sanity checks	Warning	Acknowledged
V-3JNE-VUL-009	Missing checks and validations	Warning	Acknowledged
V-3JNE-VUL-010	Maintainability issues	Warning	Fixed

5.1 Detailed Description of Issues

5.1.1 V-3JNE-VUL-001: Funds draining via malicious market creation

Severity	Critical	**Commit**	c92f1cc
Type	Data Validation	**Status**	Fixed
Location(s)	`src/Morpho.sol:149`		
Confirmed Fix At	`https://github.com/3jane-protocol/3jane-morpho-blue/pull/44`, 0666b75		

Description The `MarkdownManager` contract is responsible for a user's markdown amount based on their position, and `CreditLine` contract is responsible for credit line creation, validation, and configuration management. It is identified that it is possible to drain the `MorphoCredit` `waUSDC` wallet balance by creating a malicious market with attacker-controlled `CreditLine` and `MarkdownManager` contracts.

The vulnerability arises because `MorphoCredit` allows arbitrary market creation, even though all borrowing operations are ultimately funded from a shared `waUSDC` wallet.

By crafting a `MarkdownManager` that manipulates return values during defaulted loan accounting, an attacker can artificially inflate a market's `totalSupplyAssets` without providing any real deposits. This manipulated state enables malicious borrowers to bypass normal collateralization checks and borrow directly against the protocol's wallet balance. As a result, real `waUSDC` tokens can be withdrawn from the protocol even though the market has never received actual liquidity.

Attack Sequence

Normally, assets cannot be borrowed from distinct markets because each market supply is accounted separately, and only increased when funds are supplied to the market directly. Borrows are always checked to be less than a market's current supply, which preventing cross-market borrows in usual scenarios. However, this sequence shows how to increase the number of assets recorded in an attacker-created market's supply without actually supplying any assets.

1. Deploy a malicious `MarkdownManager` that implements `calculateMarkdown()` function to return: `N` on the first call, 0 on the second call. The value `N` can be arbitrary and correlates to how much the attacker could drain from Morpho.
2. Deploy a malicious `CreditLine` contract that is tied to the created `MarkdownManager` .
3. Register a new market that points to the newly created CreditLine in the MorphoBlue using `createMarket()`
4. Call `creditLine.setCreditLine(attacker, HIGH_COLLATERAL_AMOUNT)` to signal that the attacker user has enough collateral to borrow from the market.
5. Attacker borrows `10^6 - 1` shares from the newly created Market. This succeeds since these are virtual shares and they do not mint a single Asset, all checks in the `borrow()` pass.

6. Even though the attacker did not take any real assets, the accounting system will round up the assets corresponding to that share amount and this will give `1` asset. This is important, since all borrower premiums and penalties will wind up on this value.
7. From the malicious CreditLine, call `MorphoCredit.closeCycleAndPostObligations()` with some suitable parameters (e.g. `repaymentBps` = `10^4` and `endingBalance` = `10^6`). This will register a loan in the system. From now on, the interest starts to accrue for this borrower. After some time, if the loan is not paid out, it will be switched to Default state. This is crucial since the vulnerable logic only triggers for Defaulted loans.
8. After the Grace period and Delinquent period + 1 day, call the `MorphoCredit.accrueBorrowerPremium(id, BORROWER)`. This will add a penalty to `totalBorrowAssets` of the market. This will also trigger the `MarkdownManager` for the first time.
9. Some time later, make another call to `MorphoCredit.accrueBorrowPremium(id, BORROWER)`. This will trigger the `MarkdownManager` for the second time. Due to how the accounting is implemented for Defaulted loans, the market's `totalSupplyAsset` will be increased on the value returned by the `MarkdownManager` which is controlled by the attacker.
10. Using another address, `attacker2` , set another credit line for themselves with a huge credit limit.
11. Attacker user `attacker2` borrows everything the MoprhoCredit waUSDC wallet has. This will work since the malicious market is now accounted as being rich, the only check in borrow will successfully pass.

Impact This vulnerability can allow an attacker to create fraudulent markets that appear funded despite containing no real deposits. Once inflated through the malicious accounting logic, these markets permit borrowers to drain the `MorphoCredit` `waUSDC` wallet. Since this wallet backs all investor funds supplied through `USD3`, exploitation would lead to the complete loss of protocol-held assets. The issue therefore represents a **critical protocol-draining risk** with direct financial loss to depositors and a threat to the solvency of the system.

Recommendation It is recommended to ensure that access control mechanisms are implemented when registering a new market.

Developers Response The developers have been notified and have responded with a fix.

5.1.2 V-3JNE-VUL-002: Griefing attack preventing withdrawals via small donations

Severity	High	**Commit**	10f8d1d
Type	Access Control	**Status**	Fixed
Location(s)	src/ ▶ USD3.sol:427 ▶ sUSD3.sol:170		
Confirmed Fix At	https://github.com/3jane-protocol/usd3/pull/2, 42cd24e		

Description The USD3 and sUSD3 contracts allow anyone to make a deposit "on behalf of" a receiver. When such a deposit occurs, the `receiver`'s lock timer is reset, preventing withdrawals until the lock period expires (up to 90 days).

```
// Each deposit extends commitment for entire balance
if (minCommitmentTime > 0) {
    depositTimestamp[receiver] = block.timestamp;
}
```

An attacker can exploit this by sending a minimal deposit (just above zero) to a victim's address. This small deposit is enough to reset the withdrawal lock, and the attack can be repeated before each lock expiry to keep the victim permanently locked.

Impact A malicious user can exploit this mechanism to prevent legitimate users from withdrawing by repeatedly resetting their lock period with tiny deposits. This effectively locks the victim's funds.

Recommendation It is recommended to implement checks to ensure that no arbitrary user can extent an unsuspecting user's withdraw lock period maliciously.

Developers Response The developers now track whitelisted depositors and only extend the withdraw lock period if the depositor is in the whitelist or is the receiver.

5.1.3 V-3JNE-VUL-003: USD3/sUSD3 lock period bypass via uncontrolled token transfers

Severity	High	**Commit**	10f8d1d
Type	Logic Error	**Status**	Fixed
Location(s)	`src/` ▶ `USD3.sol` ▶ `sUSD3.sol`		
Confirmed Fix At	`https://github.com/3jane-protocol/usd3/pull/1`, 4160262		

Description It was identified that the current implementation of the `USD3` and `sUSD3` contracts allows bypassing of the commitment/lock-down period through unrestricted token transfers. The vulnerability arises because the `startCooldown()` function can be called by any address regardless of whether it has previously deposited. An attacker can exploit this by using two addresses under their control: one to hold the actual `sUSD3` shares and another to prematurely initiate the cool-down. Once the cool-down has expired for the second address, the attacker can transfer shares from the first address and withdraw immediately, effectively circumventing the intended lock period.

Attack sequence for sUSD3

1. User deposits their USDC into sUSD3, they receive sUSD3 shares in return.
2. User2 calls `sUSD3.startCooldown(2**256-1)` . This will start the cool-down period for User2, because the check `block.timestamp >= lockedUntil[msg.sender]` always holds for the user who never deposited anything.
3. After the cool-down period is over, User makes a transfer of their sUSD3 shares to User2
4. User2 should be able to withdraw them , since `availableWithdrawLimit()` doesn't check if the `lockedUntil[owner]` is non-zero.

A similar scenario allows to bypass USD3 commitment period.

Impact This vulnerability can allow an attacker to bypass the enforced lock or commitment period, undermining a critical mechanism designed to restrict withdrawals. As a result, malicious users can gain an unfair advantage over other users.

Recommendation It is recommended to implement more robust mechanisms for tracking a user's commitment period.

Developers Response The developers have provided a fix for this issue.

Veridise Response The developers provided a fix for this issue even before it was discovered. However, the attack scenario was discovered independently.

5.1.4 V-3JNE-VUL-004: Denial-of-service due to borrowed shares variable manipulation

Severity	Medium	**Commit**	c92f1cc
Type	Data Validation	**Status**	Fixed
Location(s)	`src/Morpho.sol:233`		
Confirmed Fix At	`https://github.com/3jane-protocol/3jane-morpho-blue/pull/46`, ffd7f32		

Description The `borrow()` function in `Morpho.sol` contains a flaw that allows borrowers to obtain borrow shares without receiving any underlying assets when the market is empty.

Specifically, an attacker can borrow in amounts such as `VIRTUAL_SHARES - 1`, `2 * VIRTUAL_SHARES - 1`, `4 * VIRTUAL_SHARES - 1`, and so on-resulting in shares with no assets backing them. By repeatedly exploiting this, the attacker can artificially inflate the `totalBorrowShares` variable.

Impact This manipulation can cause `totalBorrowShares` to grow arbitrarily large, leading to a denial-of-service condition when the value is cast to `uint128` .

Legitimate borrowers would be unable to borrow assets without triggering an overflow or other safety checks, effectively halting protocol lending functionality.

Recommendation It is recommended to enforce a check ensuring that borrowers cannot acquire shares unless at least one unit of the underlying asset is borrowed.

Developers Response The developers addressed the issue by introducing a validation check in `Morpho.sol` to prevent borrowing shares that would result in zero assets. Specifically, the function now reverts with a newly defined `InsufficientBorrowAmount` error if `assets == 0` while `shares > 0`.

5.1.5 V-3JNE-VUL-005: Storage slots indexes mismatch

Severity	Medium	**Commit**	c92f1cc
Type	Data Validation	**Status**	Fixed
Location(s)	`src/libraries/periphery/` ▶ `MorphoCreditStorageLib.sol:16-22` ▶ `MorphoStorageLib.sol:20-22`		
Confirmed Fix At	`https://github.com/3jane-protocol/3jane-morpho-blue/pull/45`, b697027		

Description In **MorphoStorageLib**, the library references a non-existent `isAuthorized` mapping at slot `6`. This variable does not exist in `Morpho.sol`, resulting in invalid storage access. Additionally, the slot assignment for `nonce` and `idToMarketParams` was incorrect, with `nonce` mapped to slot `7` instead of `6` and `idToMarketParams` mapped to slot `8` instead of `7`.

```
    uint256 internal constant IS_AUTHORIZED_SLOT = 6;
    uint256 internal constant NONCE_SLOT = 7;
    uint256 internal constant ID_TO_MARKET_PARAMS_SLOT = 8;
```

In **MorphoCreditStorageLib**, the `HELPER_SLOT` constant was incorrectly set to `20`. Since the base `Morpho` contract occupies slots `0-17`, the `Helper` must reside at slot `19`, not `20`. Furthermore, the library incorrectly referenced `protocolConfig` at slot `21`, even though it is an immutable variable and does not occupy storage. These errors shifted subsequent slot definitions by two positions, breaking all functions in the `idTo.*` family.

```
    // MorphoCredit storage starts at slot 20 (after Morpho base storage and gap)
    uint256 internal constant HELPER_SLOT = 20;
    uint256 internal constant PROTOCOL_CONFIG_SLOT = 21;
    uint256 internal constant USD3_SLOT = 22;
    uint256 internal constant BORROWER_PREMIUM_SLOT = 23;
    uint256 internal constant PAYMENT_CYCLE_SLOT = 24;
    uint256 internal constant REPAYMENT_OBLIGATION_SLOT = 25;
    uint256 internal constant MARKDOWN_STATE_SLOT = 26;
```

Impact These libraries are not directly used by Morpho Blue, and are meant as helpers that integrators can reuse or adapts to their own needs. the identified storage slot mismatch can result in incorrect variable reads/writes, broken functionality, and potential contract failure.

Recommendation It is recommended to remove `isAuthorizedand` `protocolConfig`, and update the remaining variables to represent the appropriate storage slots.

Developers response The developers have implemented the recommendation.

5.1.6 V-3JNE-VUL-006: Callback onMorphoSupply is called before token transfer

Severity	Warning	**Commit**	c92f1cc
Type	Logic Error	**Status**	Acknowledged
Location(s)	`src/Morpho.sol:191`		
Confirmed Fix At	N/A		

Description When calling `Morpho.supply()`, the caller can provide optional arbitrary data to pass to the `onMorphoSupply` callback for the caller. It was identified that this call is made **before** the tokens have been transferred, potentially leading to confusion if the caller expects the contract balance to be already updated at the time of the call.

Impact This behavior can cause logic errors or misinterpretation in integrator contracts relying on `onMorphoSupply`. Such confusion could lead to incorrect accounting, unsafe assumptions in downstream logic, or subtle integration bugs.

While no reentrancies were identified, this may still provide an avenue for reentrancy attacks in further upgrades of the protocol or for third-party protocol integrators.

Recommendation Execute the callback after tokens have been transferred. Alternatively, if this is intended behavior, update the comments/docs to clearly reflect this in order to avoid confusion.

Developers Response The developers chose not to address the issue since it originates from the parent protocol, Morpho Blue, which follows the same pattern.

5.1.7 V-3JNE-VUL-007: Missing event emissions for important state changes

Severity	Warning	**Commit**	c92f1cc
Type	Maintainability	**Status**	Acknowledged
Location(s)	`src/CreditLine.sol:79-112`		
Confirmed Fix At	N/A		

Description The following functions modify critical state variables but do not emit an event notifying it:

1. **src/CreditLine.sol**
   - `setOzd` @L79
   - `setMm` @ L90
   - `setProver` @ L102
   - `setInsuranceFund` @ L112

Impact Without event emissions, off-chain services, monitoring systems, or auditors may be unable to track critical state changes. This reduces transparency, complicates debugging, and can hinder incident response since important updates may go unnoticed.

Recommendation It is recommended to add event emissions when functions modify important states.

Developers Response The developers acknowledged the issue and decided not to provide a fix.

5.1.8 V-3JNE-VUL-008: Missing sanity checks

Severity	Warning	**Commit**	c92f1cc
Type	Maintainability	**Status**	Acknowledged
Location(s)	`src/` ► `Morpho.sol:138` ► `MorphoCredit.sol:748` ► `ProtocolConfig.sol:66`		
Confirmed Fix At	N/A		

Description It is identified that many setters do not implement sanity checks before assigning new values to sensitive state variables.

1. **src/ProtocolConfig.sol @ L66**
   - `setConfig` : It is recommended to introduce sanity checks for some or all config values. Since they play a crucial role in the protocol's stability, a single error could cost a fortune.
2. **src/Morpho.sol @ L138**
   - `setFeeRecipient` : It is recommended to implement a check to ensure that the new fee recipient is not set to an invalid address.
3. **src/MorphoCredit.sol @ L748**
   - `_updateBorrowerMarkdown` : The `IMarkdownManager` interface provides a function called `isValidForMarket(id)` which checks if a market is valid for markdown calculations. It is not called before calculating a new markdown.

Impact Incorrectly setting and updating values for sensitive states can lead to the protocol not functioning properly until proper values are passed in again, causing temporary denial of service and wasted gas fees.

Developers Response Developers have acknowledged the issue and decided not to provide a fix.

5.1.9 V-3JNE-VUL-009: Missing checks and validations

Severity	Warning	**Commit**	10f8d1d
Type	Maintainability	**Status**	Acknowledged
Location(s)	`src/USD3.sol:573-605`		
Confirmed Fix At	N/A		

Description It is identified that many setters do not implement sanity checks before assigning new values to sensitive state variables.

1. **src/USD3.sol**
   a) `setWhiteListEnabled` @ L573
   b) `setMinCommitmentTime` @ L605
   c) `setMinDeposit` @ L593
   d) `setWhiteList` @ L581

Impact Incorrectly setting and updating values for sensitive states can lead to the protocol not functioning properly until proper values are passed in again. Setting the current value as the new value may cause a waste of time and gas fees.

Recommendation It is recommended to implement sanity checks to ensure that the new values are valid and different than the current values.

Developers Response Developers do not plan to provide a fix for this.

5.1.10 V-3JNE-VUL-010: Maintainability issues

Severity	Warning	**Commit**	c92f1cc
Type	Maintainability	**Status**	Fixed
Location(s)	`src/Morpho.sol`		
Confirmed Fix At	`https://github.com/3jane-protocol/3jane-morpho-blue/pull/47/`, 00517a3		

Description During the security review, several maintainability issues were discovered.

1. The following were declared within the project source code but never utilized:
   - **src/Morpho.sol @ L58**
     - `nonce`
   - **src/interfaces/IMorphoCallbacks.sol @ L26**
     - `IMorphoFlashLoanCallback`
   - **src/irm/adaptive-curve-irm/interfaces/IAaveMarket.sol @ L69**
     - `IAaveMarket`
   - **src/irm/adaptive-curve-irm/libraries/ConstantsLib.sol @ L7**
     - `ConstantsLib`
   - **src/interfaces/IMorpho.sol @ L8**
     - `collateralToken`
   - **src/libraries/EventsLib.sol**
     - `FlashLoan` @ L104
     - `SetAuthorization`@ L111
     - `IncrementNonce`@ L119
     - `MarkdownManagerSet` @ L188
   - **src/libraries/ErrorsLib.sol**
     - `HealthyPosition` @ L85
     - `InvalidSignature` @ L88
     - `signatureExpired` @ L91
     - `InvalidNonce` @ L94
     - `InvalidMarkdownManager` @ L139
2. Function `setCreditLine` has a misleading name, as it does not configure the overall market credit line but instead sets the credit limit for a specific borrower, and also sets the PremiumRate for the borrower.
3. Library `MorphoStorageLib` does not implement a getter function for `totalMarkDownAmount`.
4. The following functions calls return a value, which is not being checked or used by the contract source code.
   - **src/CreditLine.sol @ L218**
     - `IERC20(marketParams.loanToken).approve(address(MORPHO), cover);`
   - **src/Helper.sol @ L48 - L51**
     - `IERC20(USDC).approve(WAUSDC, type(uint256).max);`
     - `IERC20(WAUSDC).approve(USD3, type(uint256).max);`

- `IERC20(WAUSDC).approve(MORPHO, type(uint256).max);`
- `IERC20(USD3).approve(sUSD3, type(uint256).max);`

► **src/Morpho.sol @ L161**

- `IMorphoSupplyCallback(msg.sender).onMorphoSupply(assets, data)`

Impact Although the above issues are maintainability issues with no security impact, leaving them unfixed will increase the likelihood of future bugs and increase the difficulty of extending the code base.

Recommendation Address the minor issues reported above.

Developers Response Developers addressed most of the observations.