Audit Reports

[High-1]

[Title] Cross-chain Replay in borrowAsset() & swapToBorrow()

[Severity] High

[Description] Cross-chain Replay: If an User borrows assets on Chain A, they could reuse the same signature (generated from the same data) on Chain B, assuming the signature is valid for the same borrow hash structure on both chains.

[Proof of concept] Admin signs signature with his private key with information about the collateral address, borrow address, collateral amount, borrow amount, address of the User to use signature and nonce.

Chain A User calls borrowAsset() borrows assets. Chain B User reuses the same signedHash to borrow assets.

[Mitigation]

```
bytes32 _borrowHash = keccak256(
    abi.encode(
        collaterals,
        borrows,
        collateralAmounts,
        borrowsAmounts,
        msg.sender,
        nonceOf[msg.sender],
        block.chainid // Include the chainId in the hash to make it unique per
chain
    )
);
```

[High-2]

[Title] First Depositor Issue in depositAsset()

[Severity] High

[Description] The depositAsset function is vulnerable to the first depositor bug, where the first depositor intentionally gain an unfair advantage due to how shares (_share) are calculated for initial deposits.

Example: Bob wants to deposit 10 WETH Alice sees Bob's transaction in the mempool and frontruns it (Alice becomes the first depositor). She deposits just 1 wei. Bob deposits the 10 WETH but due to rounding issues he receives 0 shares. Alice calls `removeAsset()` and gets 10 WETH + 1 wei withdrawed to her since he owns the 100% of the shares.

[Medium-1]

[Title] Missing time limit for signature

[Severity] Medium

[Description]

Some actions in financial contracts are time-sensitive, and allowing a signature to be valid indefinitely could break expected business rules. For instance, if the contract terms change or the collateral value fluctuates, an old signature might lead to unfair borrowing conditions or outdated agreements.

[Proof of Concept]

Example: A user signs a borrowing agreement for certain collateral amounts and borrowing amounts today. Later, the market conditions change, and collateral values drop or borrowing values gets too high , making the original agreement unfair for the contract owner. By adding a time limit (e.g., valid for 1 hour), the signature would become invalid once the time expires, ensuring that the agreement can only be executed within the intended window

[Mitigation]

```
bytes32 _borrowHash = keccak256(
    abi.encode(
        collaterals,
        borrows,
        collateralAmounts,
        borrowsAmounts,
        msg.sender,
        nonceOf[msg.sender],
        deadline // Add expiry
    )
);

// Check if the signature is expired
if (block.timestamp > deadline) {
    revert SignatureExpired();
}
```

[Medium-2]

[Title] Missing Approval when transfering

[Severity] Medium

[Description] In borrowAsset() when user is trying to send collateral it will fail since there is missing approval. Same applies for in repayLoan().

[Mitigation]

Add approve for contract to use the tokens of the user.

[Medium-3]

[Title] Using transferFrom instead of transfer

[Severity] Medium

[Description] In borrowAssets() when contract is trying to send borrows to user it incorrectly uses `transferFrom` instead of `transfer` when transferring borrow tokens from the contract to the user. This implementation error will cause the function to revert due to missing allowances. same applies for in repayLoan() when trying to send collateral back to user.

[Mitigation]

```
token.safeTransfer(msg.sender, _amount); //use this when transfering to user.
```

[Medium-4]

[Title] Use safeTransfer & safeTransferFrom

[Severity] Medium

[Description] Some ERC-20 tokens will return on failure instead of reverting a transaction, Some tokens will even not return any value. not all tokens adhere to the standart of ERC-20

[Mitigation] Consider adding OpenZeppelin SafeERC20.

[Informational-1]

[Title] Ensure Collateral Addresses and Amounts Arrays Have Matching Lengths

[Severity] Informational

[Description] The lengths of the collaterals and collateralAmounts arrays are not explicitly checked within the function. If their lengths differ, this can lead to mismatched indexing, potentially causing unexpected behavior or errors when processing collateral addresses and their corresponding amounts.

This issue assumes the off-chain logic ensures both arrays are correctly populated. However, without a validation check in the contract, there's no guarantee of alignment, which could result in incorrect collateral handling.

[Mitigation]

```
if (collaterals.lenght != collateralAmounts.lenght) revert
```

[Informational-2]

[Title] Redundant Processed Hash Check in borrowAsset Function

[Severity] Informational

[Description] Summary: The processed[_borrowHash] check in the borrowAsset function is redundant because the nonceOf[msg.sender] mechanism already prevents signature reuse. Reusing a signature with an outdated

nonce will fail the isAdmin verification, as the recomputed _borrowHash will not match the original signed hash.

[Mitigation] Consider removing the processed[_borrowHash] mapping and its associated logic to simplify the code and reduce gas costs.

[Informational-3]

[Title] Gas Optimization: Revert Early if _share is Zero in removeAsset

[Severity] Informational

[Description] In the removeAsset function, if the _share of msg.sender is zero, the function still proceeds to execute and consumes unnecessary gas before returning. Instead of allowing the transaction to continue and waste gas, it would be more efficient to revert the transaction early when _share is found to be zero. This prevents further execution and optimizes gas usage.

[Mitigation]

```
if(_share == 0) revert
```

[Informational-4]

[Title] Underflow Revert: Add Check for Excess Withdrawal in removeAsset

[Severity] Informational

[Description]

In the removeAsset function, if a user attempts to withdraw more shares than they hold, the transaction will fail, but the reason for failure may not be immediately clear to the user. This is due to the absence of a check to verify if shareToWithdraw exceeds the balance of shareOf[msg.sender][token]. Since shareOf[address][token] is a uint256 and cannot be negative (with a minimum value of 0), the transaction fails when the withdrawal amount is too large, but the failure message is not informative. Adding a condition to explicitly check and revert the transaction with a meaningful message would improve clarity and user experience.

[Mitigation]

```
if (shareToWithdraw > shareOf[msg.sender][token]) revert
```

[Informational-5]

[Title] Consider adding deadline for swap

[Severity] Informational

[Description]

In the Mooniswap swap function, users can specify a minAmount parameter to protect against slippage, ensuring that the output of the trade meets a minimum threshold. However, the function does not include a deadline parameter, which means users can set a high minAmount and wait indefinitely for the market to meet favorable conditions. This behavior could lead to inefficiencies in the protocol, as transactions could be left pending for extended periods without completing.

[Mitigation]

Consider adding deadline for swap