MythX

REPORT 5F6C8980492A590019ABC23D

Created	Thu Sep 24 2020 11:56:48 GMT+0000 (Coordinated Universal Time)
Number of analyses	30
User	mueller.berndt11@gmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
64072149-95e1-4d05-81d5-e4208ec04a6a	contracts/configuration/LendingPoolAddressesProvider.sol	5
aa0471eb-41a9-425c-919f-c039e9bec79b	contracts/configuration/LendingPoolAddressesProviderRegistry.sol	5
787b5a58-ab84-4997-9769-1a9cb250789b	contracts/flashloan/base/FlashLoanReceiverBase.sol	2
aa5c3fc8-e82a-49b4-b7ec-5b974c474ba9	contracts/lendingpool/DefaultReserveInterestRateStrategy.sol	5
0d66d682-635b-4584-9cef-c06f9aebc452	contracts/lendingpool/LendingPool.sol	6
fcb20a27-ea34-4c2c-8337-4b6080c86416	contracts/lendingpool/LendingPoolCollateralManager.sol	6
23ab2d22-b300-4bc8-92c4-416d329e1f38	contracts/lendingpool/LendingPoolConfigurator.sol	18
1f29afcc-4ca0-44d5-b5b1-e18baf5fe1a9	contracts/lendingpool/LendingPoolStorage.sol	2
cdda6dbe-d81a-44ef-bb74-3aa6875cf4af	contracts/libraries/configuration/ReserveConfiguration.sol	2
c9046ec5-f746-4833-95d3-cf9748f0eaaa	contracts/libraries/configuration/UserConfiguration.sol	1
5dace3c5-59c8-4c59-837a-23c372e566f3	contracts/libraries/helpers/Errors.sol	1
3c3b9416-2983-4992-a084-7a564e270296	contracts/libraries/helpers/Helpers.sol	8
6022a28e-9b87-4987-a9d6-dc17290fb2b8	contracts/libraries/logic/GenericLogic.sol	2
e071d582-e0a5-465c-b949-4edb02623c9d	contracts/libraries/logic/ReserveLogic.sol	2
b5723ab5-6725-4541-af0b-f07c760417c4	contracts/libraries/logic/ValidationLogic.sol	4
70bf2cfb-9d4c-408f-a452-6a400561d106	contracts/libraries/math/MathUtils.sol	1
e56dd809-edd6-403a-9954-4eb8cb298588	math/PercentageMath.sol	1
3656f064-b54a-4a01-ad11-908d3da38b1b	contracts/libraries/math/SafeMath.sol	0
2251e105-ace4-4c41-a907-827f212451ec	math/WadRayMath.sol	1
852fe9ee-1457-4089-b3a2-19874a1c1f2a	contracts/misc/AaveProtocolTestHelpers.sol	5

22d0c0ba-6cb6-4fdf-8fc4-8da514a01ed8	contracts/misc/Address.sol	0
bfd056ae-a898-484d-9996-0d6be2220d53	contracts/misc/ChainlinkProxyPriceProvider.sol	3
7b69a08c-ed60-46fa-88e6-af537fd929fb	contracts/misc/IERC20DetailedBytes.sol	1
0890c604-4286-4b95-a6ef-1bd45ebbdd69	misc/SafeERC20.sol	0
587fd904-4ae8-4f58-a36f-381ae9fb99f1	contracts/misc/WalletBalanceProvider.sol	5
5e9b5f59-7cbd-4fb7-adb3-f43704ea2a31	contracts/tokenization/AToken.sol	59
4dcb9ffa-3855-4e7d-af3a-584ef34ba7c4	tokenization/IncentivizedERC20.sol	3
0c813c52-8d60-46ab-bfca-838632e2f8dd	contracts/tokenization/StableDebtToken.sol	7
c735bc1c-fa4a-4e89-9725-ceabbd2138bc	contracts/tokenization/VariableDebtToken.sol	8
48db364a-404d-4867-afab-71437ab170c0	contracts/tokenization/base/DebtTokenBase.sol	8

Analysis 64072149-95e1-4d05-81d5-e4208ec04a6a MythX

Started	
Finished	Thu Sep 24 2020 11:58:18 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Configuration/LendingPoolAddressesProvider.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	4

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

contracts/configuration/LendingPoolAddressesProvider.sol

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/configuration/LendingPoolAddressesProvider.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {Ownable} from '@openzeppelin/contracts/access/Ownable.sol';
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

Analysis aa0471eb-41a9-425c-919f-c039e9bec79b MythX

Started	
Finished	Thu Sep 24 2020 11:58:21 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Configuration/LendingPoolAddressesProviderRegistry.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	4

ISSUES

MEDIUM

SWC-128

Loop over unbounded data structure.

Gas consumption in function "_addToAddressesProvidersList" in contract "LendingPoolAddressesProviderRegistry" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`contracts/configuration/LendingPoolAddressesProviderRegistry.sol`

Locations

```
75  **/
76  function _addToAddressesProvidersList(address provider) internal {
77  for (uint256 i = 0; i < addressesProvidersList.length; i++) {
78  if (addressesProvidersList[i] == provider) {
79  return;
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/configuration/LendingPoolAddressesProviderRegistry.sol`

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  import {Ownable} from '@openzeppelin/contracts/access/Ownable.sol';
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "addressesProviders" is internal. Other possible visibility settings are public and private.

Source file

`contracts/configuration/LendingPoolAddressesProviderRegistry.sol`

Locations

```
15
16 contract LendingPoolAddressesProviderRegistry is Ownable, ILendingPoolAddressesProviderRegistry {
17 mapping(address => uint256) addressesProviders;
18 address[] addressesProvidersList;
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "addressesProvidersList" is internal. Other possible visibility settings are public and private.

Source file

`contracts/configuration/LendingPoolAddressesProviderRegistry.sol`

Locations

```
16 contract LendingPoolAddressesProviderRegistry is Ownable, ILendingPoolAddressesProviderRegistry {
17 mapping(address => uint256) addressesProviders;
18 address[] addressesProvidersList;
19
20 /**
```

LOW

SWC-128

Loop over unbounded data structure.

Gas consumption in function "getAddressesProvidersList" in contract "LendingPoolAddressesProviderRegistry" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`contracts/configuration/LendingPoolAddressesProviderRegistry.sol`

Locations

```
41 address[] memory activeProviders = new address[](maxLength);
42
43 for (uint256 i = 0; i < addressesProvidersList.length; i++) {
44 if (addressesProviders[addressesProvidersList[i]] > 0) {
45 activeProviders[i] = addressesProvidersList[i];
```

Analysis 787b5a58-ab84-4997-9769-1a9cb250789b MythX

Started	
Finished	Thu Sep 24 2020 11:58:24 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Flashloan/Base/FlashLoanReceiverBase.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	2

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/flashloan/base/FlashLoanReceiverBase.sol`

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

LOW

SWC-131

Unused state variable "_addressesProvider".

The state variable "_addressesProvider" is declared within the contract "FlashLoanReceiverBase" but its value does not seem to be used anywhere.

Source file

`contracts/flashloan/base/FlashLoanReceiverBase.sol`

Locations

```
13 using SafeMath for uint256;
14
15 ILendingPoolAddressesProvider internal _addressesProvider;
16
17 constructor(ILendingPoolAddressesProvider provider) public {
```

Analysis aa5c3fc8-e82a-49b4-b7ec-5b974c474ba9 MythX

Started	
Finished	Thu Sep 24 2020 11:58:29 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Lendingpool/DefaultReserveInterestRateStrategy.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	4

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/lendingpool/DefaultReserveInterestRateStrategy.sol`

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

Analysis `0d66d682-635b-4584-9cef-c06f9aebc452` MythX

Started	Thu Sep 24 2020 11:58:50 GMT+0000 (Coordinated Universal Time)
Finished	Thu Sep 24 2020 12:44:07 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Lendingpool/LendingPool.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
1	0	5

ISSUES

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

`contracts/libraries/math/MathUtils.sol`

Locations

```
56  }
57
58  uint256 expMinusOne = exp - 1;
59
60  uint256 expMinusTwo = exp > 2 ? exp - 2 : 0;
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3  pragma experimental ABIEncoderV2;
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
717
718 return (
719 IERC20(asset).balanceOf(reserve.aTokenAddress),
720 IERC20(reserve.stableDebtTokenAddress).totalSupply(),
721 IERC20(reserve.variableDebtTokenAddress).totalSupply(),
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis fcb20a27-ea34-4c2c-8337-4b6080c86416 MythX

Started	Thu Sep 24 2020 11:58:50 GMT+0000 (Coordinated Universal Time)
Finished	Thu Sep 24 2020 12:44:05 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Lendingpool/LendingPoolCollateralManager.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	6

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/lendingpool/LendingPoolCollateralManager.sol

Locations

```
// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.6.8;

import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

contracts/libraries/helpers/Helpers.sol

Locations

```
{
return (
DebtTokenBase(reserve.stableDebtTokenAddress).balanceOf(user),
DebtTokenBase(reserve.variableDebtTokenAddress).balanceOf(user)
);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/libraries/helpers/Helpers.sol`

Locations

```
return (
DebtTokenBase(reserve.stableDebtTokenAddress).balanceOf(user),
DebtTokenBase(reserve.variableDebtTokenAddress).balanceOf(user)
);
}
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```

function _beforeTokenTransfer(
address from,
address to,
uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
function _beforeTokenTransfer(
address from,
address to,
uint256 amount
) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis 23ab2d22-b300-4bc8-92c4-416d329e1f38 MythX

Started	
Finished	Thu Sep 24 2020 11:58:49 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Lendingpool/LendingPoolConfigurator.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	18

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.6.8;
pragma experimental ABIEncoderV2;
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
**/
function setPoolPause(bool val) external onlyAaveAdmin {
pool.setPause(val);
}
}
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
550 onlyAaveAdmin
551 {
552 pool.setReserveInterestRateStrategyAddress(asset, rateStrategyAddress);
553 emit ReserveInterestRateStrategyChanged(asset, rateStrategyAddress);
554 }
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
355 **/
356 function disableReserveAsCollateral(address asset) external onlyAaveAdmin {
357 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
358
359 currentConfig.setLtv(0);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
369 **/
370 function enableReserveStableRate(address asset) external onlyAaveAdmin {
371 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
372
373 currentConfig.setStableRateBorrowingEnabled(true);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
318 **/
319 function disableBorrowingOnReserve(address asset) external onlyAaveAdmin {
320 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
321
322 currentConfig.setBorrowingEnabled(false);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
456 **/
457 function unfreezeReserve(address asset) external onlyAaveAdmin {
458 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
459
460 currentConfig.setFrozen(false);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
502 **/
503 function setLiquidationThreshold(address asset, uint256 threshold) external onlyAaveAdmin {
504 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
505
506 currentConfig.setLiquidationThreshold(threshold);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
383 **/
384 function disableReserveStableRate(address asset) external onlyAaveAdmin {
385 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
386
387 currentConfig.setStableRateBorrowingEnabled(false);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
397 **/
398 function activateReserve(address asset) external onlyAaveAdmin {
399 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
400
401 currentConfig.setActive(true);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
517 **/
518 function setLiquidationBonus(address asset, uint256 bonus) external onlyAaveAdmin {
519 ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);
520
521 currentConfig.setLiquidationBonus(bonus);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
**/
function freezeReserve(address asset) external onlyAaveAdmin {
ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);

currentConfig.setFrozen(true);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
**/
function setReserveDecimals(address asset, uint256 decimals) external onlyAaveAdmin {
ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);

currentConfig.setDecimals(decimals);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
**/
function setLtv(address asset, uint256 ltv) external onlyAaveAdmin {
ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);

currentConfig.setLtv(ltv);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
**/
function setReserveFactor(address asset, uint256 reserveFactor) external onlyAaveAdmin {
ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);

currentConfig.setReserveFactor(reserveFactor);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
uint256 liquidationBonus
) external onlyAaveAdmin {
ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);

currentConfig.setLtv(ltv);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPoolConfigurator.sol`

Locations

```
onlyAaveAdmin
{
ReserveConfiguration.Map memory currentConfig = pool.getConfiguration(asset);

currentConfig.setBorrowingEnabled(true);
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

contracts/lendingpool/LendingPoolConfigurator.sol

Locations

```
197 function initialize(ILendingPoolAddressesProvider provider) public initializer {
198 addressesProvider = provider;
199 pool = ILendingPool(addressesProvider.getLendingPool());
200 }
```

Analysis 1f29afcc-4ca0-44d5-b5b1-e18baf5fe1a9 MythX

Started	
Finished	Thu Sep 24 2020 11:58:54 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Lendingpool/LendingPoolStorage.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	1

ISSUES

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

contracts/libraries/logic/ReserveLogic.sol

Locations

```
133 {
134 require(
135 ReserveLogic.InterestRateMode.STABLE == ReserveLogic.InterestRateMode(interestRateMode) ||
136 ReserveLogic.InterestRateMode.VARIABLE == ReserveLogic.InterestRateMode(interestRateMode),
137 Errors.INVALID_INTEREST_RATE_MODE_SELECTED
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/lendingpool/LendingPoolStorage.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {UserConfiguration} from '../libraries/configuration/UserConfiguration.sol';
```

Analysis cdda6dbe-d81a-44ef-bb74-3aa6875cf4af MythX

Started	
Finished	Thu Sep 24 2020 11:58:59 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Configuration/ReserveConfiguration.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	1

ISSUES

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

contracts/libraries/logic/ReserveLogic.sol

Locations

```
133 {
134 require(
135 ReserveLogic.InterestRateMode.STABLE == ReserveLogic.InterestRateMode(interestRateMode) ||
136 ReserveLogic.InterestRateMode.VARIABLE == ReserveLogic.InterestRateMode(interestRateMode),
137 Errors.INVALID_INTEREST_RATE_MODE_SELECTED
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/libraries/configuration/ReserveConfiguration.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

Analysis c9046ec5-f746-4833-95d3-cf9748f0eaaa MythX

Started	
Finished	Thu Sep 24 2020 11:59:03 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Configuration/UserConfiguration.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/libraries/configuration/UserConfiguration.sol

Locations

```
// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.6.8;

import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

Analysis 5dace3c5-59c8-4c59-837a-23c372e566f3 MythX

Started	
Finished	Thu Sep 24 2020 11:59:05 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Helpers/Errors.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/libraries/helpers/Errors.sol

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  /**
```

Analysis 3c3b9416-2983-4992-a084-7a564e270296 MythX

Started	
Finished	Thu Sep 24 2020 11:59:08 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Libraries/Helpers/Helpers.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	7

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/libraries/helpers/Helpers.sol`

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {DebtTokenBase} from '../../tokenization/base/DebtTokenBase.sol';
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis 6022a28e-9b87-4987-a9d6-dc17290fb2b8 MythX

Started	
Finished	Thu Sep 24 2020 11:59:12 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Logic/GenericLogic.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	1

ISSUES

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

contracts/libraries/logic/ReserveLogic.sol

Locations

```
133 {
134 require(
135 ReserveLogic.InterestRateMode.STABLE == ReserveLogic.InterestRateMode(interestRateMode) ||
136 ReserveLogic.InterestRateMode.VARIABLE == ReserveLogic.InterestRateMode(interestRateMode),
137 Errors.INVALID_INTEREST_RATE_MODE_SELECTED
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/libraries/logic/GenericLogic.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3 pragma experimental ABIEncoderV2;
```

Analysis e071d582-e0a5-465c-b949-4edb02623c9d MythX

Started	
Finished	Thu Sep 24 2020 11:59:16 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Logic/ReserveLogic.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	1

ISSUES

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

contracts/libraries/logic/ReserveLogic.sol

Locations

```
133 {
134 require(
135 ReserveLogic.InterestRateMode.STABLE == ReserveLogic.InterestRateMode(interestRateMode) ||
136 ReserveLogic.InterestRateMode.VARIABLE == ReserveLogic.InterestRateMode(interestRateMode),
137 Errors.INVALID_INTEREST_RATE_MODE_SELECTED
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/libraries/logic/ReserveLogic.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

Analysis b5723ab5-6725-4541-af0b-f07c760417c4 MythX

Started	
Finished	Thu Sep 24 2020 11:59:21 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Libraries/Logic/ValidationLogic.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	4

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/libraries/logic/ValidationLogic.sol`

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3 pragma experimental ABIEncoderV2;
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis 70bf2cfb-9d4c-408f-a452-6a400561d106 MythX

Started	
Finished	Thu Sep 24 2020 11:59:26 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Math/MathUtils.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/libraries/math/MathUtils.sol

Locations

```
// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.6.8;

import {SafeMath} from '@openzeppelin/contracts/math/SafeMath.sol';
```

Analysis e56dd809-edd6-403a-9954-4eb8cb298588 MythX

Started	
Finished	Thu Sep 24 2020 11:59:28 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Math/PercentageMath.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`math/PercentageMath.sol`

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  import {Errors} from '../helpers/Errors.sol';
```

Analysis 3656f064-b54a-4a01-ad11-908d3da38b1b MythX

Started	
Finished	Thu Sep 24 2020 11:59:30 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Libraries/Math/SafeMath.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

Analysis 2251e105-ace4-4c41-a907-827f212451ec MythX

Started	
Finished	Thu Sep 24 2020 11:59:32 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Math/WadRayMath.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

math/WadRayMath.sol

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  import {Errors} from '../helpers/Errors.sol';
```

Analysis 852fe9ee-1457-4089-b3a2-19874a1c1f2a MythX

Started	
Finished	Thu Sep 24 2020 11:59:34 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Misc/AaveProtocolTestHelpers.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	4

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

contracts/configuration/LendingPoolAddressesProvider.sol

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/misc/AaveProtocolTestHelpers.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3 pragma experimental ABIEncoderV2;
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

Analysis 22d0c0ba-6cb6-4fdf-8fc4-8da514a01ed8 MythX

Started	
Finished	Thu Sep 24 2020 11:59:37 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Misc/Address.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

Analysis bfd056ae-a898-484d-9996-0d6be2220d53 MythX

Started	
Finished	Thu Sep 24 2020 11:59:39 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Misc/ChainlinkProxyPriceProvider.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	3

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/misc/ChainlinkProxyPriceProvider.sol`

Locations

```
// SPDX-License-Identifier: agpl-3.0
pragma solidity ^0.6.8;

import {Ownable} from '@openzeppelin/contracts/access/Ownable.sol';
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/misc/ChainlinkProxyPriceProvider.sol`

Locations

```
// If there is no registered source for the asset, call the fallbackOracle
if (address(source) == address(0)) {
return _fallbackOracle.getAssetPrice(asset);
} else {
int256 price = IChainlinkAggregator(source).latestAnswer();
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

contracts/misc/ChainlinkProxyPriceProvider.sol

Locations

```
80 // If there is no registered source for the asset, call the fallbackOracle
81 if (address(source) == address(0)) {
82 return _fallbackOracle.getAssetPrice(asset);
83 } else {
84 int256 price = IChainlinkAggregator(source).latestAnswer();
```

Analysis 7b69a08c-ed60-46fa-88e6-af537fd929fb MythX

Started	
Finished	Thu Sep 24 2020 11:59:41 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Misc/IERC20DetailedBytes.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/misc/IERC20DetailedBytes.sol

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  contract IERC20DetailedBytes {
```

Analysis 0890c604-4286-4b95-a6ef-1bd45ebbdd69 MythX

Started	
Finished	Thu Sep 24 2020 11:59:43 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Misc/SafeERC20.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

Analysis 587fd904-4ae8-4f58-a36f-381ae9fb99f1 MythX

Started	
Finished	Thu Sep 24 2020 11:59:45 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Misc/WalletBalanceProvider.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	4

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/misc/WalletBalanceProvider.sol`

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {Address} from '@openzeppelin/contracts/utils/Address.sol';
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

Analysis 5e9b5f59-7cbd-4fb7-adb3-f43704ea2a31 MythX

Started	Thu Sep 24 2020 12:00:00 GMT+0000 (Coordinated Universal Time)
Finished	Thu Sep 24 2020 12:45:27 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Tokenization/AToken.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
4	4	51

ISSUES

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

contracts/libraries/math/MathUtils.sol

Locations

```
56  }
57
58  uint256 expMinusOne = exp - 1;
59
60  uint256 expMinusTwo = exp > 2 ? exp - 2 : 0;
```

HIGH

SWC-112

The contract delegates execution to another contract with a user-supplied address.

The smart contract delegates execution to a user-supplied address.This could allow an attacker to execute arbitrary code in the context of this contract account and manipulate the state of the contract account or execute actions on its behalf.

Source file

contracts/lendingpool/LendingPool.sol

Locations

```

//solium-disable-next-line
(bool success, bytes memory result) = collateralManager.delegatecall(
abi.encodeWithSignature(
'liquidationCall(address,address,address,uint256,bool)',
collateral,
asset,
user,
purchaseAmount,
receiveAToken
)
);
require(success, Errors.LIQUIDATION_CALL_FAILED);
```

HIGH

SWC-112

The contract delegates execution to another contract with a user-supplied address.

The smart contract delegates execution to a user-supplied address.This could allow an attacker to execute arbitrary code in the context of this contract account and manipulate the state of the contract account or execute actions on its behalf.

Source file

contracts/lendingpool/LendingPool.sol

Locations

```

//solium-disable-next-line
(bool success, bytes memory result) = collateralManager.delegatecall(
abi.encodeWithSignature(
'repayWithCollateral(address,address,address,uint256,address,bytes)',
collateral,
principal,
user,
principalAmount,
receiver,
params
)
);
require(success, Errors.FAILED_REPAY_WITH_COLLATERAL);
```

HIGH

SWC-112

The contract delegates execution to another contract with a user-supplied address.

The smart contract delegates execution to a user-supplied address.This could allow an attacker to execute arbitrary code in the context of this contract account and manipulate the state of the contract account or execute actions on its behalf.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
620
621 //solium-disable-next-line
622 (bool success, bytes memory result) = collateralManager.delegatecall(
623 abi.encodeWithSignature(
624 'swapLiquidity(address,address,address,uint256,bytes)',
625 receiverAddress,
626 fromAsset,
627 toAsset,
628 amountToSwap,
629 params
630 )
631 );
632 require(success, Errors.FAILED_COLLATERAL_SWAP);
```

MEDIUM

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
523 }
524
525 _flashLiquidationLocked = false;
526 }
```

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
317 (uint256 stableDebt, uint256 variableDebt) = Helpers.getUserCurrentDebt(msg.sender, reserve);
318
319 ReserveLogic.InterestRateMode interestRateMode = ReserveLogic.InterestRateMode(rateMode);
320
321 ValidationLogic.validateSwapRateMode(
```

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
261 (uint256 stableDebt, uint256 variableDebt) = Helpers.getUserCurrentDebt(onBehalfOf, reserve);
262
263 ReserveLogic.InterestRateMode interestRateMode = ReserveLogic.InterestRateMode(rateMode);
264
265 //default to max amount
```

MEDIUM

SWC-110

An assertion violation was triggered.

It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
904
905 if (
906 ReserveLogic.InterestRateMode(vars.interestRateMode) == ReserveLogic.InterestRateMode.STABLE
907 ) {
908 currentStableRate = reserve.currentStableBorrowRate;
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/tokenization/AToken.sol`

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {IncentivizedERC20} from './IncentivizedERC20.sol';
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
176 ) external override view returns (uint256) {
177 return
178 _borrowAllowance[_reserves[asset].getDebtTokenAddress(interestRateMode)][fromUser][toUser];
179 }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
195 address debtToken = _reserves[asset].getDebtTokenAddress(interestRateMode);
196
197 _borrowAllowance[debtToken][msg.sender][user] = amount;
198 emit BorrowAllowanceDelegated(asset, msg.sender, user, interestRateMode, amount);
199 }
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
428 );
429
430 _usersConfig[msg.sender].setUsingAsCollateral(reserve.id, useAsCollateral);
431
432 if (useAsCollateral) {
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/libraries/configuration/UserConfiguration.sol`

Locations

```
46 ) internal {
47 self.data =
48 (self.data & ~(1 << (reserveIndex * 2 + 1))) |
49 (uint256(_usingAsCollateral ? 1 : 0) << (reserveIndex * 2 + 1));
50 }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/libraries/configuration/UserConfiguration.sol`

Locations

```
45  bool _usingAsCollateral
46  ) internal {
47  self.data =
48  (self.data & ~(1 << (reserveIndex * 2 + 1))) |
49  (uint256(_usingAsCollateral ? 1 : 0) << (reserveIndex * 2 + 1));
50  }
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
950  function _addReserveToList(address asset) internal {
951  bool reserveAlreadyAdded = false;
952  require(_reservesList.length < MAX_NUMBER_RESERVES, Errors.NO_MORE_RESERVES_ALLOWED);
953  for (uint256 i = 0; i < _reservesList.length; i++)
954  if (_reservesList[i] == asset) {
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
956  }
957  if (!reserveAlreadyAdded) {
958  _reserves[asset].id = uint8(_reservesList.length);
959  _reservesList.push(asset);
960  }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
956 }
957 if (!reserveAlreadyAdded) {
958 _reserves[asset].id = uint8(_reservesList.length);
959 _reservesList.push(asset);
960 }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
957 if (!reserveAlreadyAdded) {
958 _reserves[asset].id = uint8(_reservesList.length);
959 _reservesList.push(asset);
960 }
961 }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
327 );
328
329 reserve.updateState();
330
331 if (interestRateMode == ReserveLogic.InterestRateMode.STABLE) {
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
331 if (interestRateMode == ReserveLogic.InterestRateMode.STABLE) {
332 //burn stable rate tokens, mint variable rate tokens
333 IStableDebtToken(reserve.stableDebtTokenAddress).burn(msg.sender, stableDebt);
334 IVariableDebtToken(reserve.variableDebtTokenAddress).mint(
335 msg.sender,
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
339 } else {
340 //do the opposite
341 IVariableDebtToken(reserve.variableDebtTokenAddress).burn(
342 msg.sender,
343 variableDebt,
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
342 msg.sender,
343 variableDebt,
344 reserve.variableBorrowIndex
345 );
346 IStableDebtToken(reserve.stableDebtTokenAddress).mint(
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
331 if (interestRateMode == ReserveLogic.InterestRateMode.STABLE) {
332 //burn stable rate tokens, mint variable rate tokens
333 IStableDebtToken(reserve.stableDebtTokenAddress).burn(msg.sender, stableDebt);
334 IVariableDebtToken(reserve.variableDebtTokenAddress).mint(
335 msg.sender,
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
332 //burn stable rate tokens, mint variable rate tokens
333 IStableDebtToken(reserve.stableDebtTokenAddress).burn(msg.sender, stableDebt);
334 IVariableDebtToken(reserve.variableDebtTokenAddress).mint(
335 msg.sender,
336 stableDebt,
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
335 msg.sender,
336 stableDebt,
337 reserve.variableBorrowIndex
338 );
339 } else {
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
339 } else {
340 //do the opposite
341 IVariableDebtToken(reserve.variableDebtTokenAddress).burn(
342 msg.sender,
343 variableDebt,
344 reserve.variableBorrowIndex
345 );
346 IStableDebtToken(reserve.stableDebtTokenAddress).mint(
347 msg.sender,
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
344 reserve.variableBorrowIndex
345 );
346 IStableDebtToken(reserve.stableDebtTokenAddress).mint(
347 msg.sender,
348 variableDebt,
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
347 msg.sender,
348 variableDebt,
349 reserve.currentStableBorrowRate
350 );
351 }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
332 //burn stable rate tokens, mint variable rate tokens
333 IStableDebtToken(reserve.stableDebtTokenAddress).burn(msg.sender, stableDebt);
334 IVariableDebtToken(reserve.variableDebtTokenAddress).mint(
335 msg.sender,
336 stableDebt,
337 reserve.variableBorrowIndex
338 );
339 } else {
340 //do the opposite
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
351 }
352
353 reserve.updateInterestRates(asset, reserve.aTokenAddress, 0, 0);
354
355 emit Swap(asset, msg.sender);
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
344 reserve.variableBorrowIndex
345 );
346 IStableDebtToken(reserve.stableDebtTokenAddress).mint(
347 msg.sender,
348 variableDebt,
349 reserve.currentStableBorrowRate
350 );
351 }
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
351 }
352
353 reserve.updateInterestRates(asset, reserve.aTokenAddress, 0, 0);
354
355 emit Swap(asset, msg.sender);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
419 ReserveLogic.ReserveData storage reserve = _reserves[asset];
420
421 ValidationLogic.validateSetUseReserveAsCollateral(
422 reserve,
423 asset,
424 _reserves,
425 _usersConfig[msg.sender],
426 _reservesList,
427 _addressesProvider.getPriceOracle()
428 );
429
430 _usersConfig[msg.sender].setUsingAsCollateral(reserve.id, useAsCollateral);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
816 ) external override {
817 _onlyLendingPoolConfigurator();
818 _reserves[asset].init(
819 aTokenAddress,
820 stableDebtAddress,
821 variableDebtAddress,
822 interestRateStrategyAddress
823 );
824 _addReserveToList(asset);
825 }
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
457
458 //solium-disable-next-line
459 (bool success, bytes memory result) = collateralManager.delegatecall(
460 abi.encodeWithSignature(
461 'liquidationCall(address,address,address,uint256,bool)',
462 collateral,
463 asset,
464 user,
465 purchaseAmount,
466 receiveAToken
467 )
468 );
469 require(success, Errors.LIQUIDATION_CALL_FAILED);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
504
505 //solium-disable-next-line
506 (bool success, bytes memory result) = collateralManager.delegatecall(
507 abi.encodeWithSignature(
508 'repayWithCollateral(address,address,address,uint256,address,bytes)',
509 collateral,
510 principal,
511 user,
512 principalAmount,
513 receiver,
514 params
515 )
516 );
517 require(success, Errors.FAILED_REPAY_WITH_COLLATERAL);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
718 return (
719 IERC20(asset).balanceOf(reserve.aTokenAddress),
720 IERC20(reserve.stableDebtTokenAddress).totalSupply(),
721 IERC20(reserve.variableDebtTokenAddress).totalSupply(),
722 reserve.currentLiquidityRate,
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/libraries/helpers/Helpers.sol`

Locations

```
24 return (
25 DebtTokenBase(reserve.stableDebtTokenAddress).balanceOf(user),
26 DebtTokenBase(reserve.variableDebtTokenAddress).balanceOf(user)
27 );
28 }
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
620
621 //solium-disable-next-line
622 (bool success, bytes memory result) = collateralManager.delegatecall(
623 abi.encodeWithSignature(
624 'swapliquidity(address,address,address,uint256,bytes)',
625 receiverAddress,
626 fromAsset,
627 toAsset,
628 amountToSwap,
629 params
630 )
631 );
632 require(success, Errors.FAILED_COLLATERAL_SWAP);
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/logic/ReserveLogic.sol`

Locations

```
86
87  //solium-disable-next-line
88  if (timestamp == uint40(block.timestamp)) {
89  //if the index was updated in the same block, no need to perform any calculation
90  return reserve.liquidityIndex;
91  }
92
93  uint256 cumulated = MathUtils
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/logic/ReserveLogic.sol`

Locations

```
109
110 //solium-disable-next-line
111 if (timestamp == uint40(block.timestamp)) {
112 //if the index was updated in the same block, no need to perform any calculation
113 return reserve.variableBorrowIndex;
114 }
115
116 uint256 cumulated = MathUtils
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`node_modules/@openzeppelin/contracts/math/SafeMath.sol`

Locations

```
59  */
60  function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
61  require(b <= a, errorMessage);
62  uint256 c = a - b;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/MathUtils.sol`

Locations

```
52 uint256 exp = block.timestamp.sub(uint256(lastUpdateTimestamp));
53
54 if (exp == 0) {
55 return WadRayMath.ray();
56 }
57
58 uint256 expMinusOne = exp - 1;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/MathUtils.sol`

Locations

```
58 uint256 expMinusOne = exp - 1;
59
60 uint256 expMinusTwo = exp > 2 ? exp - 2 : 0;
61
62 uint256 ratePerSecond = rate / SECONDS_PER_YEAR;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
156 function wadToRay(uint256 a) internal pure returns (uint256) {
157 uint256 result = a * WAD_RAY_RATIO;
158 require(result / WAD_RAY_RATIO == a, Errors.MULTIPLICATION_OVERFLOW);
159 return result;
160 }
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`node_modules/@openzeppelin/contracts/math/SafeMath.sol`

Locations

```
79  // benefit is lost if 'b' is also tested.
80  // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
81  if (a == 0) {
82  return 0;
83  }
84
85  uint256 c = a * b;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`node_modules/@openzeppelin/contracts/math/SafeMath.sol`

Locations

```
84
85  uint256 c = a * b;
86  require(c / a == b, "SafeMath: multiplication overflow");
87
88  return c;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`node_modules/@openzeppelin/contracts/math/SafeMath.sol`

Locations

```
84
85  uint256 c = a * b;
86  require(c / a == b, "SafeMath: multiplication overflow");
87
88  return c;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
127 uint256 result = a * RAY;
128
129 require(result / RAY == a, Errors.MULTIPLICATION_OVERFLOW);
130
131 result += halfB;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
131 result += halfB;
132
133 require(result >= halfB, Errors.ADDITION_OVERFLOW);
134
135 return result / b;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`node_modules/@openzeppelin/contracts/math/SafeMath.sol`

Locations

```
29 function add(uint256 a, uint256 b) internal pure returns (uint256) {
30 uint256 c = a + b;
31 require(c >= a, "SafeMath: addition overflow");
32
33 return c;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
99  **/
100 function rayMul(uint256 a, uint256 b) internal pure returns (uint256) {
101 if (a == 0) {
102 return 0;
103 }
104
105 uint256 result = a * b;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
105 uint256 result = a * b;
106
107 require(result / a == b, Errors.MULTIPLICATION_OVERFLOW);
108
109 result += halfRAY;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
105 uint256 result = a * b;
106
107 require(result / a == b, Errors.MULTIPLICATION_OVERFLOW);
108
109 result += halfRAY;
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`contracts/libraries/math/WadRayMath.sol`

Locations

```
109 result += halfRAY;
110
111 require(result >= halfRAY, Errors.ADDITION_OVERFLOW);
112
113 return result / RAY;
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/lendingpool/LendingPool.sol`

Locations

```
717
718 return (
719 IERC20(asset).balanceOf(reserve.aTokenAddress),
720 IERC20(reserve.stableDebtTokenAddress).totalSupply(),
721 IERC20(reserve.variableDebtTokenAddress).totalSupply(),
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis 4dcb9ffa-3855-4e7d-af3a-584ef34ba7c4 MythX

Started	
Finished	Thu Sep 24 2020 11:59:58 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Tokenization/IncentivizedERC20.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	3

ISSUES

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

tokenization/IncentivizedERC20.sol

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

tokenization/IncentivizedERC20.sol

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis `0c813c52-8d60-46ab-bfca-838632e2f8dd` MythX

Started	
Finished	Thu Sep 24 2020 12:00:01 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	`Contracts/Tokenization/StableDebtToken.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
1	0	6

ISSUES

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

`contracts/libraries/math/MathUtils.sol`

Locations

```
56  }
57
58  uint256 expMinusOne = exp - 1;
59
60  uint256 expMinusTwo = exp > 2 ? exp - 2 : 0;
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/tokenization/StableDebtToken.sol`

Locations

```
1  // SPDX-License-Identifier: agpl-3.0
2  pragma solidity ^0.6.8;
3
4  import {Context} from '@openzeppelin/contracts/GSN/Context.sol';
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "_timestamps" is internal. Other possible visibility settings are public and private.

Source file

`contracts/tokenization/StableDebtToken.sol`

Locations

```
21
22 uint256 private _avgStableRate;
23 mapping(address => uint40) _timestamps;
24 uint40 _totalSupplyTimestamp;
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "_totalSupplyTimestamp" is internal. Other possible visibility settings are public and private.

Source file

`contracts/tokenization/StableDebtToken.sol`

Locations

```
22 uint256 private _avgStableRate;
23 mapping(address => uint40) _timestamps;
24 uint40 _totalSupplyTimestamp;
25
26 constructor(
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis c735bc1c-fa4a-4e89-9725-ceabbd2138bc MythX

Started	
Finished	Thu Sep 24 2020 12:00:06 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Tokenization/VariableDebtToken.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	7

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

contracts/configuration/LendingPoolAddressesProvider.sol

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/tokenization/VariableDebtToken.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {Context} from '@openzeppelin/contracts/GSN/Context.sol';
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```

Analysis 48db364a-404d-4867-afab-71437ab170c0 MythX

Started	
Finished	Thu Sep 24 2020 12:00:14 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.21
Main Source File	Contracts/Tokenization/Base/DebtTokenBase.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	7

ISSUES

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

contracts/configuration/LendingPoolAddressesProvider.sol

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.6.8"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/tokenization/base/DebtTokenBase.sol

Locations

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity ^0.6.8;
3
4 import {Context} from '@openzeppelin/contracts/GSN/Context.sol';
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
137 } else {
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
132 if (proxyAddress == address(0)) {
133 proxy = new InitializableAdminUpgradeabilityProxy();
134 proxy.initialize(newAddress, address(this), params);
135 _addresses[id] = address(proxy);
136 emit ProxyCreated(id, address(proxy));
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/configuration/LendingPoolAddressesProvider.sol`

Locations

```
136 emit ProxyCreated(id, address(proxy));
137 } else {
138 proxy.upgradeToAndCall(newAddress, params);
139 }
140 }
```

LOW

SWC-131

Unused function parameter "from".

The value of the function parameter "from" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
247
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
```

LOW

SWC-131

Unused function parameter "to".

The value of the function parameter "to" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
248 function _beforeTokenTransfer(
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
```

LOW

SWC-131

Unused function parameter "amount".

The value of the function parameter "amount" for the function "_beforeTokenTransfer" of contract "IncentivizedERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Source file

`contracts/tokenization/IncentivizedERC20.sol`

Locations

```
249 address from,
250 address to,
251 uint256 amount
252 ) internal virtual {}
253 }
```