CERTIK

Aave Protocol V2

Security Assessment

December 2nd, 2020

For :
Aave Protocol V2

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Camden Smallwood @ CertiK
camden.smallwood@certik.org

Sheraz Arshad @ CertiK
sheraz.arshad@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Aave Protocol V2
Description	The next version of the AAVE DeFi protocol, supporting the lending, burrowing and flash-loaning of crypto assets.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. f756f44a8d6a328cd545335e46e7128939db88c4 2. 57ee9e0a7cf9a41f969965d4f49eefa907a30dfe 3. 58c326e8e40d62ec7bdbc75f7467e05e930d0e04 4. 750920303e33b66bc29862ea3b85206dda9ce786

Audit Summary

Delivery Date	December 2nd, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	3
Timeline	September 28th, 2020 - December 2nd, 2020

Vulnerability Summary

Total Issues	45
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	3
Total Informational	42

Executive Summary

The codebase of the project comprises Aave's V2 Lending Protocol implementation. The protocol implementation enables depositors to provide liquidity to the protocol and earn a passive income proportionate to their deposits whereas borrowers are able to utilize the deposited capital by borrowing it in an overcollateralized or undercollateralized fashion.

The upgraded version of the protocol utilizes a similar AToken system to the first version and supports multi-collateral flash loans with a percentage based fee. Certain mathematical formulas are also defined within the code that are meant to implement the formulas detailed in the whitepaper of Aave, including linear and compound interest.

We validated these formulas as well as rounding adaptations made to commonly used libraries such as `WadRayMath`.

The protocol optimizes its configuration parameters via bitwise operations, enabling a single data bit to represent a boolean flag. We validated that the bitwise operands utilized by the `ReserveConfiguration` implementation of the said single bit representation. The protocol utilizes an adaptation of the Open Zeppelin proxy pattern whereby a versioning system was introduced. We verified that the implementation was sound and does not deviate from the standard.

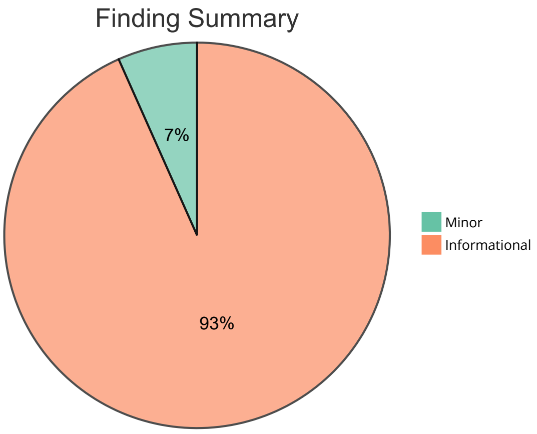
Overall, no serious vulnerabilities were observed in the codebase and the code itself was well readable and developed conforming to the latest standards in Solidity.

Files In Scope

ID	Contract	Location
LPA	LendingPoolAddressesProvider.sol	contracts/configuration/LendingPoolAddressesProvider.sol
LPP	LendingPoolAddressesProviderRegistry.sol	contracts/configuration/LendingPoolAddressesProviderRegistry.sol
FLR	FlashLoanReceiverBase.sol	contracts/flashloan/base/FlashLoanReceiverBase.sol
IFL	IFlashLoanReceiver.sol	contracts/flashloan/interfaces/IFlashLoanReceiver.sol
IAI	IAaveIncentivesController.sol	contracts/interfaces/IAaveIncentivesController.sol
ICA	IChainlinkAggregator.sol	contracts/interfaces/IChainlinkAggregator.sol
IEA	IExchangeAdapter.sol	contracts/interfaces/IExchangeAdapter.sol
ILP	ILendingPool.sol	contracts/interfaces/ILendingPool.sol
ILA	ILendingPoolAddressesProvider.sol	contracts/interfaces/ILendingPoolAddressesProvider.sol
ILR	ILendingPoolAddressesProviderRegistry.sol	contracts/interfaces/ILendingPoolAddressesProviderRegistry.sol
ILO	ILendingRateOracle.sol	contracts/interfaces/ILendingRateOracle.sol
IPO	IPriceOracle.sol	contracts/interfaces/IPriceOracle.sol
IPG	IPriceOracleGetter.sol	contracts/interfaces/IPriceOracleGetter.sol
IRI	IReserveInterestRateStrategy.sol	contracts/interfaces/IReserveInterestRateStrategy.sol
IUE	IUniswapExchange.sol	contracts/interfaces/IUniswapExchange.sol
DRI	DefaultReserveInterestRateStrategy.sol	contracts/lendingpool/DefaultReserveInterestRateStrategy.sol
LPL	LendingPool.sol	contracts/lendingpool/LendingPool.sol
LPC	LendingPoolCollateralManager.sol	contracts/lendingpool/LendingPoolCollateralManager.sol
LEN	LendingPoolConfigurator.sol	contracts/lendingpool/LendingPoolConfigurator.sol
LPS	LendingPoolStorage.sol	contracts/lendingpool/LendingPoolStorage.sol
BIA	BaseImmutableAdminUpgradeabilityProxy.sol	contracts/libraries/aave-upgradeability/BaseImmutableAdminUpgradeabilityProxy.sol
IIA	InitializableImmutableAdminUpgradeabilityProxy.sol	contracts/libraries/aave-upgradeability/InitializableImmutableAdminUpgradeabilityProxy.sol
VIE	VersionedInitializable.sol	contracts/libraries/aave-upgradeability/VersionedInitializable.sol
RCN	ReserveConfiguration.sol	contracts/libraries/configuration/ReserveConfiguration.sol
UCN	UserConfiguration.sol	contracts/libraries/configuration/UserConfiguration.sol
ERR	Errors.sol	contracts/libraries/helpers/Errors.sol
HEL	Helpers.sol	contracts/libraries/helpers/Helpers.sol
SLB	StringLib.sol	contracts/libraries/helpers/StringLib.sol
GLC	GenericLogic.sol	contracts/libraries/logic/GenericLogic.sol
RLC	ReserveLogic.sol	contracts/libraries/logic/ReserveLogic.sol
VLC	ValidationLogic.sol	contracts/libraries/logic/ValidationLogic.sol
MUS	MathUtils.sol	contracts/libraries/math/MathUtils.sol
PMH	PercentageMath.sol	contracts/libraries/math/PercentageMath.sol
WRM	WadRayMath.sol	contracts/libraries/math/WadRayMath.sol
ATN	AToken.sol	contracts/tokenization/AToken.sol
DTB	DebtTokenBase.sol	contracts/tokenization/base/DebtTokenBase.sol

ID	Contract	Location
DAA	DelegationAwareAToken.sol	contracts/tokenization/DelegationAwareAToken.sol
IER	IncentivizedERC20.sol	contracts/tokenization/IncentivizedERC20.sol
IAT	IAToken.sol	contracts/tokenization/interfaces/IAToken.sol
ISB	IScaledBalanceToken.sol	contracts/tokenization/interfaces/IScaledBalanceToken.sol
ISD	IStableDebtToken.sol	contracts/tokenization/interfaces/IStableDebtToken.sol
ITC	ITokenConfiguration.sol	contracts/tokenization/interfaces/ITokenConfiguration.sol
IVD	IVariableDebtToken.sol	contracts/tokenization/interfaces/IVariableDebtToken.sol
SDT	StableDebtToken.sol	contracts/tokenization/StableDebtToken.sol
VDT	VariableDebtToken.sol	contracts/tokenization/VariableDebtToken.sol

Findings



ID	Title	Type	Severity	Resolved
LPL-01	Redundant `public` Visibility	Gas Optimization	Informational	✓
LPL-02	`modifier` Over `function`	Gas Optimization	Informational	✓
LPL-03	Hardcoded Raw Function Invocation	Coding Style	Informational	✓
LPL-04	`if-revert` to `require`	Coding Style	Informational	✓
LPL-05	Redundant `receive` Implementation	Coding Style	Informational	✓
LPL-06	Redundant Variable Initialization	Coding Style	Informational	✓
LPL-07	Typo in Comment	Inconsistency	Informational	✓
LPL-08	Documentation Discrepancy	Inconsistency	Informational	✓
LPL-09	Redundant Function Implementation	Coding Style	Informational	✓
LPL-10	Incorrect Function Visibility	Coding Style	Informational	(!)✓
LEN-01	Redundant `public` Visibility	Gas Optimization	Informational	✓
LEN-02	Insufficient Input Sanitization	Logical Issue	Minor	✓
LEN-03	Documentation Discrepancy	Inconsistency	Informational	✓
LEN-04	Incorrect Function Visibility	Coding Style	Informational	(!)✓
LPC-01	Variable Tight-Packing	Coding Style	Informational	✓
LPC-02	Typo in Comment	Inconsistency	Informational	✓
LPC-03	Typo in Comment	Inconsistency	Informational	✓
LPC-04	Redundant Variable Initialization	Coding Style	Informational	(!)✓

ID	Title	Type	Severity	Resolved
LPC-05	Unused Return Value of External Call	Coding Style	Informational	✓
RLC-01	Incorrect Variable Sanitization	Logical Issue	Minor	✓
RLC-02	Documentation Discrepancy	Inconsistency	Informational	✓
RLC-03	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
SDT-01	Incorrect Variable Sanitization	Logical Issue	Minor	✓
SDT-02	Unused Imports	Coding Style	Informational	✓
LPA-01	Ineffectual Code	Gas Optimization	Informational	(!)✓
LPA-02	Unused Variables	Gas Optimization	Informational	✓
LPP-01	Visibility Specifiers Missing	Language Specific	Informational	✓
LPP-02	Redundant Variable Initialization	Gas Optimization	Informational	✓
LPP-03	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	(!)✓
LPP-04	Documentation Discrepancy	Inconsistency	Informational	✓
ILR-01	Documentation Discrepancy	Inconsistency	Informational	✓
LPS-01	Inefficient Use of Storage	Gas Optimization	Informational	(!)✓
GLC-01	Redundant Variable Initialization	Coding Style	Informational	✓
GLC-02	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	(!)✓
GLC-03	Typo in Comment	Inconsistency	Informational	(!)✓

ID	Title	Type	Severity	Resolved
GLC-04	Inefficient Code Block	Gas Optimization	Informational	!✓
VLC-01	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
BIA-01	Ineffectual File Import	Coding Style	Informational	!✓
VIE-01	Redundant Variable Initialization	Coding Style	Informational	!✓
ATN-01	Incomplete Variable Name	Coding Style	Informational	!✓
ATN-02	Ineffectual Usage of `receive` Function	Gas Optimization	Informational	✓
ATN-03	Usage of Contract Type Over Interface Type	Coding Style	Informational	✓
VDT-01	Unused Imports	Coding Style	Informational	✓
DTB-01	Unused Imports	Coding Style	Informational	✓
IER-01	Incorrect Function Visibility	Coding Style	Informational	✓

LPL-01: Redundant `public` Visibility

Type	Severity	Location
Gas Optimization	Informational	LendingPool.sol L42-L47

Description:

The linked contract-level declarations contain configuration values that do not need to be made externally visible on-chain as they are easily viewable in the source code of the contracts.

Recommendation:

We advise that the `public` specifier is omitted from these declarations to reduce bytecode size.

Alleviation:

The Aave team informed us that these variables could be important for onchain integrators of the protocol and as such, they decided to retain the `public` visibility of those variables.

LPL-02: `modifier` Over `function`

Type	Severity	Location
Gas Optimization	Informational	LendingPool.sol L49-L57, L59-L68

Description:

The linked `function` implementations act as access-control checks for certain restricted functions yet are invoked like regular functions.

Recommendation:

We advise that they are instead changed to `modifier` implementations. If code duplication is desired to be avoided due to the `modifier` code replace, we advise that `modifier`s are coded that simply invoke those functions conforming to the same pattern.

Alleviation:

The linked functions are now utilized by correspondingly named `modifier`s, thus nullifying this exhibit.

LPL-03: Hardcoded Raw Function Invocation

Type	Severity	Location
Coding Style	Informational	LendingPool.sol L429-L436

Description:

The linked `delegatecall` invokes the `liquidationCall` function from the `collateralManager` address with a set of input variables and a hard-coded function signature.

Recommendation:

We advise that the `selector` specifier is utilized from the `LendingPool` contract instead of utilizing the hard-coded value to ensure that a future update of the contract won't break the specified line and aid in the maintainability of the codebase.

Alleviation:

The team stated that they preferred to retain the signature hard-coded as is to prevent a development change of the collateral manager interface "breaking" the function signature.

LPL-04: `if-revert` to `require`

Type	Severity	Location
Coding Style	Informational	LendingPool.sol L442-L445

Description:

The linked lines conduct an `if` check prior to executing a `revert` statement.

Recommendation:

We advise that the clause is instead converted to a `require` statement. Additionally, the variable `returnMessage` is already a `string` and as such the tight-packing and casting appear to be redundant.

Alleviation:

The `if-revert` clause was changed to a simple `require` invocation as per the exhibit's recommendation.

LPL-05: Redundant `receive` Implementation

Type	Severity	Location
Coding Style	Informational	LendingPool.sol L748-L753

Description:

The contract implements a `receive` function that simple `revert`s inside.

Recommendation:

Solidity reverts by default if no `receive` function is implemented in the contract and thus, this implementation can be omitted.

Alleviation:

The `receive()` implementation was properly omitted from the codebase.

LPL-06: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	LendingPool.sol L902, L950, L952

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

No alleviations. This exhibit was only partly applicable as portion of the code was removed.

LPL-07: Typo in Comment

Type	Severity	Location
Inconsistency	Informational	LendingPool.sol L309

Description:

The comment on the aforementioned line has a typo where it is written as `borrowers can user` instead of `borrowers can use`.

Recommendation:

We recommend that the comment on aforementioned line be rectified by fixing the typo.

Alleviation:

Alleviations were applied by refactoring the comment on the aforementioned line.

LPL-08: Documentation Discrepancy

Type	Severity	Location
Inconsistency	Informational	LendingPool.sol L441

Description:

The comment on the aforementioned line has refers to the parameter `collateral` of the related function as `asset` which itself is a different parameter of the function.

Recommendation:

We recommend that the correct parameter name of function be specified in the comment. The parameter name of `L441` should be changed to `collateral`.

Alleviation:

Alleviations were applied as advised.

LPL-09: Redundant Function Implementation

Type	Severity	Location
Coding Style	Informational	LendingPool.sol L795-L797

Description:

The function `getReserves` on the aforementioned line returns the list of `reserves` by returning the storage variable `_reservesList`. There is another function with the same implementation inheritied from `LendingPoolStorage` called `getReservesList` which also returns the same storage variable `_reservesList`. Hence, the implementation of `getReserves` in `LendingPool` contract can be considered redudant and can be removed to reduce the `bytecode` footprint of the code and saving gas costs related to deployment.

Recommendation:

We recommend that the declaration of `getReserves` in `LendingPool` contract be removed and instead `getReservesList` inherited from `LendingPoolStorage` be used where needed.

Alleviation:

Alleviations were applied as advised.

LPL-10: Incorrect Function Visibility

Type	Severity	Location
Coding Style	Informational	LendingPool.sol L80

Description:

The function on the aforementioned line is declared as `public` while is never called internally within the contract. The functions which are never called internally from within the contract should have `external` visibility.

Recommendation:

We recommend that the function's visibility declared on the aforementioned line be changed to `external`.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

LEN-01: Redundant `public` Visibility

Type	Severity	Location
Gas Optimization	Informational	LendingPoolConfigurator.sol L209

Description:

The linked variable contains a user-defined getter called `getRevision` and is also specified as `public`.

Recommendation:

We advise that the `public` visibility specifier is omitted to reduce the bytecode of the contract.

Alleviation:

The `public` visibility specifier was removed from the `constant` variable.

LEN-02: Insufficient Input Sanitization

Type	Severity	Location
Logical Issue	Minor	LendingPoolConfigurator.sol L497-L510, L527-L540, L542-L555

Description:

The linked functions allow overriding of a reserve's configuration with no input sanitization.

Recommendation:

As values need to conform to certain specifications according to `configureReserveAsCollateral`, we advise similar sanitization checks are imposed in these functions to prevent misconfiguration of the protocol.

Alleviation:

The Aave team responded by stating that the sanitization aspect of the variables is conducted at the library-level and that they have introduce some extra sanitization checks outside of the library as well. Additionally, this exhibit is no longer applicable as the underlying code was relocated and revamped to not expose the same functions externally thus nullifying this exhibit.

LEN-03: Documentation Discrepancy

Type	Severity	Location
Inconsistency	Informational	LendingPoolConfigurator.sol L21

Description:

The comment on the aforementioned line refers to `LendingPool` contract as `LendingPoolCore` contract.

Recommendation:

We recommend that the comment on aforementioned line be rectified by providing correct name for the `LendingPool` contract.

Alleviation:

Alleviations were applied by refactoring the comment on the aforementioned line.

LEN-04: Incorrect Function Visibility

Type	Severity	Location
Coding Style	Informational	LendingPoolConfigurator.sol L197, L211

Description:

The functions on the aforementioned lines are declared as `public` while they are never called internally within the contract. The functions which are never called internally from within the contract should have `external` visibility.

Recommendation:

We recommend that the functions' visibility declared on the aforementioned lines be changed to `external`.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

LPC-01: Variable Tight-Packing

Type	Severity	Location
Coding Style	Informational	LendingPoolCollateralManager.sol L68

Description:

The linked variable is slotted between two `uint256` variables unoptimized.

Recommendation:

We advise that it is instead relocated under the `isCollateralEnabled` boolean to ensure tight-packing with `IAToken`. Enums are represented by the minimum `uint` type variable possible which in this case is `8`.

Alleviation:

The variables were re-ordered to properly tight-pack.

LPC-02: Typo in Comment

Type	Severity	Location
Inconsistency	Informational	LendingPoolCollateralManager.sol L132, L552

Description:

The comments on the aforementioned lines have typos where it is written as `to liquidated` instead of `to be liquidated`.

Recommendation:

We recommend that the comments on aforementioned lines be rectified by fixing the typos.

Alleviation:

Alleviations were applied by refactoring the comment on the aforementioned line.

LPC-03: Typo in Comment

Type	Severity	Location
Inconsistency	Informational	LendingPoolCollateralManager.sol L448

Description:

The comment on the aforementioned line has a typo where it is written as `Allows an user` instead of `Allows a user`.

Recommendation:

We recommend that the comment on aforementioned line be rectified by fixing the typo.

Alleviation:

The comment on the aforementioned lines was removed from the code rendering this exhibit inapplicable.

LPC-04: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	LendingPoolCollateralManager.sol L563-L564

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

LPC-05: Unused Return Value of External Call

Type	Severity	Location
Coding Style	Informational	LendingPoolCollateralManager.sol L407, L507

Description:

The aforementioned lines perform external call to `transferFrom` of `ERC20` contracts and the return value is not checked in either case. The rest of instances in the contract where an interaction with `ERC20` contract happens, the `SafeERC20` library methods are used which allows not to check the return value of the `ERC20` method call. Similarly, the aforementioned lines should use `safeTransferFrom` method from `SafeERC20` library.

Recommendation:

We recommend that the aforementioned lines should use `safeTransferFrom` method from `SafeERC20` library to perform the call.

Alleviation:

The concerned code was removed rendering this exhibit inapplicable.

RLC-01: Incorrect Variable Sanitization

Type	Severity	Location
Logical Issue	Minor	ReserveLogic.sol L169, L256-L258, L376, L388

Description:

The linked `require` statements conduct a less-than comparison with the result of `1 << 128`. This bitwise shift operation will not result in the maximum of `uint128` however.

Recommendation:

We advise that the `require` checks are corrected to use proper values for overflow checks.

Alleviation:

The linked comparisons were properly replaced with comparisons with the result of `type(uint128).max` instead of the current faulty implementation.

RLC-02: Documentation Discrepancy

Type	Severity	Location
Inconsistency	Informational	ReserveLogic.sol L102

Description:

The comment on the aforementioned line describes the behaviour of the function following it and states that with the passage of time `income is accrued`. As the function returns `debt` amount which also accrues with time, so the comment should reflect the behaviour of the function and should be rectified to `debt is accrued`.

Recommendation:

We recommend that the aforementioned part of the comment be rectified with `debt is accrued`.

Alleviation:

No alleviations were applied suggesting that the exhibit was identified incorrectly.

RLC-03: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	ReserveLogic.sol L382, L394

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

Alleviations were partly applied.

SDT-01: Incorrect Variable Sanitization

Type	Severity	Location
Logical Issue	Minor	StableDebtToken.sol L125

Description:

The linked `require` statements conduct a less-than comparison with the result of `1 << 128`. This bitwise shift operation will not result in the maximum of `uint128` however.

Recommendation:

We advise that the `require` checks are corrected to use proper values for overflow checks.

Alleviation:

As with `RLC-01`, the comparisons were updated to utilize `type(uint128).max`.

SDT-02: Unused Imports

Type	Severity	Location
Coding Style	Informational	StableDebtToken.sol L4-L6

Description:

The imports on the aforementioned lines are never used in the contract and can be safely removed without any consequences.

Recommendation:

We recommend that the unused imports on the aforementioned lines be removed.

Alleviation:

Alleviations were applied as advised.

LPA-01: Ineffectual Code

Type	Severity	Location
Gas Optimization	Informational	LendingPoolAddressesProvider.sol L127

Description:

The assignment on `L127` would be redundant when the code execution reaches `L133` as the local variable is re-assigned with a different value. This redundant assigment will result in wastages of gas which is easily avoidable by not initializing the variable `proxy` at the time of declaration. The initialization part on `L127` can be moved to `L138` within the `else` block.

Recommendation:

We recommend that the variable `proxy` be not initialized at the time of declaration to save gas cost for the code flow which jumps to the first block of `if` clause.

We recommend following changes for the code.

```
InitializableAdminUpgradeabilityProxy proxy;
bytes memory params = abi.encodeWithSignature('initialize(address)',
address(this));

if (proxyAddress == address(0)) {
    proxy = new InitializableAdminUpgradeabilityProxy();
    proxy.initialize(newAddress, address(this), params);
    _addresses[id] = address(proxy);
    emit ProxyCreated(id, address(proxy));
} else {
        proxy = InitializableAdminUpgradeabilityProxy(
      proxyAddress
        );
    proxy.upgradeToAndCall(newAddress, params);
}
```

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

LPA-02: Unused Variables

Type	Severity	Location
Gas Optimization	Informational	LendingPoolAddressesProvider.sol L22, L26, L27, L28, L31

Description:

The `constant` variables declared on the aforementioned lines are never used in the `Aave V2` codebase and hence can be removed without any consequences to save deployment gas costs.

Recommendation:

We recommend that the `constant` variables declared on the aforementioned lines be removed as they are not used in the contract.

Alleviation:

Alleviations were applied as advised.

LPP-01: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	LendingPoolAddressesProviderRegistry.sol L17-L18

Description:

The state variables on the aforementioned lines do not have visibilty specfied explicitly.

Recommendation:

We recommend that the explicit visibility of state variables on the aforementioned lines be specified.

Alleviation:

Alleviations were applied as advised.

LPP-02: Redundant Variable Initialization

Type	Severity	Location
Gas Optimization	Informational	LendingPoolAddressesProviderRegistry.sol L43, L77

Description:

The aforementioned lines declare variables of type `uint256` and initialize it with `0`. In Solidity, all un-initialized variables have a default value which for the `uint256` variable is `0`, hence the initialization part is redundant and can be removed.

Recommendation:

We recommend that the explicit initialization of type `uin256` with default value `0` be removed as it is redundant.

Alleviation:

We no longer advise to remove the default initialization of integers when used in `for` loops. This exhibit is no longer applicable.

LPP-03: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	LendingPoolAddressesProviderRegistry.sol L44, L67

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

LPP-04: Documentation Discrepancy

Type	Severity	Location
Inconsistency	Informational	LendingPoolAddressesProviderRegistry.sol L23

Description:

The comment on the aforementioned line says that the function following it should return a `boolean` value while in-fact it returns a `uint256` value which is resulting in discrepancy between the comment and the implemented code.

Recommendation:

We recommend that the comment be rectified to reflect the return of `0` for unregistered addresses providers and non-zero otherwise.

Alleviation:

The comment on the aforementioned line and the relevent code part were removed rendering this exhibit inapplicable.

ILR-01: Documentation Discrepancy

Type	Severity	Location
Inconsistency	Informational	ILendingPoolAddressesProviderRegistry.sol L6

Description:

The comment on the aforementioned line refers to `LendingPool` contract as `LendingPoolCore` contract.

Recommendation:

We recommend that the comment on aforementioned line be rectified by providing correct name for the `LendingPool` contract.

Alleviation:

The comment on the aforementioned was removed rendering this exhibit inapplicable.

LPS-01: Inefficient Use of Storage

Type	Severity	Location
Gas Optimization	Informational	LendingPoolStorage.sol L23-L24

Description:

The two `boolean` storage variables on the aforementioned lines are although packed within a single `32 bytes` slot but this slot can be entirely saved by packing these two `boolean` storage variables with the `ILendingPoolAddressesProvider _addressesProvider` storage variable on `L14`. The `ILendingPoolAddressesProvider` is an `address` type internally and occupies `20 bytes` with the rest `12 bytes` of the slot remaining unoccupied. As the `boolean` state variables need `2 bytes` for their storage and if they are declared next to `ILendingPoolAddressesProvider` then a single slot can store `ILendingPoolAddressesProvider` and both `boolean` variables resulting in saving of gas consumption related to usage of one extra slot for storage.

Recommendation:

We recommend that the `boolean` variables on the aforementioned lines be declared next to `ILendingPoolAddressesProvider _addressesProvider` so that the storage slot hosting `boolean` variables could be freed.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

GLC-01: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	GenericLogic.sol L174

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

We no longer advise to remove the default initialization of integers when used in `for` loops. This exhibit is no longer applicable.

GLC-02: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	GenericLogic.sol L219, L222

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

GLC-03: Typo in Comment

Type	Severity	Location
Inconsistency	Informational	GenericLogic.sol L258

Description:

The comment on the aforementioned line has a typo where it is written as `that an user can borrow` instead of `that a user can borrow`.

Recommendation:

We recommend that the comment on aforementioned line be rectified by fixing the typo.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

GLC-04: Inefficient Code Block

Type	Severity	Location
Gas Optimization	Informational	GenericLogic.sol L273-L278

Description:

The code block on the aforementioned lines makes use of extra local variable which can result in excessive consumption of gas which is easily avoidable. Although, in the `Aave` protocol this function is never executed as a part of transaction but any third-party contract interacting with the `Aave` protocol may call this function as a part of transaction. Additionally, it will reduce the deployment gas cost by a small number.

Recommendation:

We advise that either the expression `availableBorrowsETH.sub(borrowBalanceETH);` on `L277` be directly returned from the function or a `ternary conditional` statement can be added as follows.

```
return availableBorrowsETH < borrowBalanceETH ? 0 :
availableBorrowsETH.sub(borrowBalanceETH);
```

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

VLC-01: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	ValidationLogic.sol L40, L60, L160, L226, L229, L231, L263, L265, L306, L328, L418

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

Alleviations were partly applied.

BIA-01: Ineffectual File Import

Type	Severity	Location
Coding Style	Informational	BaseImmutableAdminUpgradeabilityProxy.sol L4

Description:

The `BaseAdminUpgradeabilityProxy` contract inherits from `BaseUpgradeabilityProxy`. The code for `BaseUpgradeabilityProxy` is brought into the context through the import of `UpgradeabilityProxy.sol`. The file `UpgradeabilityProxy.sol` itself imports `BaseUpgradeabilityProxy`. So instead of importing `UpgradeabilityProxy.sol` in `BaseAdminUpgradeabilityProxy.sol` we can directly import `BaseUpgradeabilityProxy.sol` to have the relevant code in the context.

Recommendation:

We recommend that the import of `UpgradeabilityProxy.sol` in `BaseAdminUpgradeabilityProxy.sol` be changed to `BaseUpgradeabilityProxy.sol`.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

VIE-01: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	VersionedInitializable.sol L22

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`
- `address`: All `address` types are initialized to `address(0)`
- `byte`: All `byte` types are initialized to their `byte(0)` representation
- `bool`: All `bool` types are initialized to `false`
- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

ATN-01: Incomplete Variable Name

Type	Severity	Location
Coding Style	Informational	ATokensol L26, L80

Description:

The variable name `EIP712_DOMAIN` on the aforementioned line is incomplete as it represents hash of `EIP721 Domain`. We advise that the variable name be changed to reflect the hash content of it.

Recommendation:

We recommend that the variable name be changed to `EIP712_DOMAIN_HASH`.

Alleviation:

The Aave Protocol V2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

ATN-02: Ineffectual Usage of `receive` Function

Type	Severity	Location
Gas Optimization	Informational	AToken.sol L326-L328

Description:

The `receive` function of the contract always `reverts` the transaction effectively returning any `ether` sent to the contract. The default behaviour of the contract when there is no `receive` function declared is that it returns `ether` sent to the contract and reverts the transaction. The declaration of `receive` function can be removed from the contract without any effects and it will result in reduced `bytecode` footprint of the code resulting in less gas consumption when the contract is deployed.

Recommendation:

We recommend that the declaration of `receive` function be removed to reduce the contract deployment gas cost by reducing `bytecode` footprint of the contract.

Alleviation:

Alleviations were applied as advised.

ATN-03: Usage of Contract Type Over Interface Type

Type	Severity	Location
Coding Style	Informational	AToken.sol L37

Description:

The variable type on the aforementioned line is `LendingPool` which can be changed to `ILendingPool` to comply with the practices in the rest of the code.

Recommendation:

We recommend that the variable type on the aforementioned line be changed to `ILendingPool`.

Alleviation:

Alleviations were applied as advised.

VDT-01: Unused Imports

Type	Severity	Location
Coding Style	Informational	VariableDebtToken.sol L4-L6

Description:

The imports on the aforementioned lines are never used in the contract and can be safely removed without any consequences.

Recommendation:

We recommend that the unused imports on the aforementioned lines be removed.

Alleviation:

Alleviations were applied as advised.

DTB-01: Unused Imports

Type	Severity	Location
Coding Style	Informational	DebtTokenBase.sol L4-L6

Description:

The imports on the aforementioned lines are never used in the contract and can be safely removed without any consequences.

Recommendation:

We recommend that the unused imports on the aforementioned lines be removed.

Alleviation:

Alleviations were applied as advised.

IER-01: Incorrect Function Visibility

Type	Severity	Location
Coding Style	Informational	IncentivizedERC20.sol L43, L50, L57, L64, L71, L81, L93, L108, L120, L141, L152

Description:

The functions on the aforementioned lines are declared as `public` while they are never called internally within the contract. The functions which are never called internally from within the contract should have `external` visibility.

Recommendation:

We recommend that the functions' visibility declared on the aforementioned lines be changed to `external`.

Alleviation:

We longer suggest to change the visibility of a function from `public` to `external` unless it has at least one parameter of array type. This exhibit is not applicable.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.