CERTIK

# CertiK Audit Report for Aave

# Contents

## Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Aave (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

## About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable.  For more information: https://certik.io.

# Executive Summary

This report has been prepared for **AAVE** to discover issues and vulnerabilities in the source code of their **multi-EIP compliant AAVE Token and LEND to AAVE Migrator** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

# Testing Summary

SECURITY LEVEL


Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by AAVE.

This audit was conducted to discover issues and vulnerabilities in the source code of AAVE's multi-EIP Compliant AAVE token and AAVE to LEND Migrator.

| | |
|---|---|
| TYPE | Smart Contract |
| SOURCE CODE | https://gitlab.com/aave-tech/aave-token/ |
| PLATFORM | EVM |
| LANGUAGE | Solidity |
| REQUEST DATE | July 10, 2020 |
| DELIVERY DATE | July 22, 2020 |
| METHODS | A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review. |

# Review Notes

## Introduction

CertiK team was contracted by the AAVE team to audit the design and implementation of their AAVE token smart contract, its compliance with the multiple EIPs it is meant to implement as well as its LEND to AAVE migrator contract.

The audited source code link is:

- Token & Migrator Source Code: https://gitlab.com/aave-tech/aave-token/

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

The AAVE team swiftly dealt with and responded to all of the Exhibits we laid out in the first round of audit, providing us with a new commit to continue the audit from as well as a dedicated branch for resolving the vulnerability detailed in Exhibit 9.

A final revision of the codebase was conducted on another commit hash to make sure the branch was merged properly and to ensure that all Exhibits have been uniformly dealt with or responded to. The SHA-1 digests of each file in scope can be observed in the paragraph below.

These digests were generated by running the “git ls-files -s” command for each file and copying the generated SHA-1 digest. The reason behind choosing this method of generation is the OS agnostic nature of the generated digest as a conventional SHA-1 digest would differ between OSes due to the way the newline character is represented.

## File Digests

| Filename | Hash | Pass |
| --- | --- | --- |
| AaveToken.sol | • SHA-1: 59e910fd59514a379bc0dfd9eb030821b7624411 | ✔ |
| LendToAave Migrator.sol | • SHA-1: e316261f318659c6af36bed651ae52bb026f0c49 | ✔ |
| Versioned Initializable.sol | • SHA-1: e5a8b87b8f89b6c5f28b25be9f7499d14b5b6ff3 | ✔ |
| ERC20.sol (changes detailed in README in scope) | • SHA-1: c543bae32c32b503212dc0186e792b3e7e64e5fd | ✔ |

## Documentation

The sources of truth regarding the operation of the contracts in scope were comprehensive and **greatly aided our efforts to audit the project as well as generally increase the legibility of the codebase**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the AAVE team or reported an issue.

## Summary

The codebase of the project is a typical EIP20 implementation with additional support for EIP712 and EIP2612. Additionally, as the contract is meant to be deployed via a proxy mechanism, the EIP being used is EIP1967.

**Certain optimization steps** that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however **1 minor and 1 medium severity vulnerability were identified during our audit that solely concern the specification**. The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and **can be deemed to be of high security and quality**.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however the exploitation surface is minimal and relates to a type of phishing attack against the variable sequence a user / application expects.

## Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.

## Conclusion

Our findings were either fully assimilated in the codebase or were omitted justifiably by the AAVE team in a very short timeframe, indicating **a dedicated and well-versed team** in the Solidity space **capable of maintaining a secure and exemplary codebase**.

# Findings

## Exhibit 1

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unlocked Compiler Version | Language Specific | Informational | token/AaveToken.sol: L1<br>token/LendToAaveMigrator.sol: L1 |

**[INFORMATIONAL] Description:**

The smart contract “pragma” statements regarding the compiler version indicate that version 0.6.10 or higher should be utilized.

**Recommendations:**

We advise that the compiler version is locked at version 0.6.10 or whichever Solidity version higher than that satisfies the requirements of the codebase as an unlocked compiler version can lead to discrepancies between compilations of the same source code due to compiler bugs and differences.

**Alleviation:**

As per our recommendation, the AAVE team locked both contracts at version 0.6.10 aiding in pinpointing compiler bugs should they occur.

# Exhibit 2

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Function & Variable Visibility | Coding Style | Informational | utils/VersionedInitializable.sol: L39<br>token/AaveToken.sol: L31, L121 - L123<br>token/LendToAaveMigrator.sol: L20, L74 - L76 |

**[INFORMATIONAL] Description:**

The Aave team has applied an adjusted version of the Initializable trait defined in the OpenZeppelin libraries whereby a “revision” number is utilized for discerning between initialize deployments.

To achieve this, a “getRevision” function is meant to be implemented as an “internal” function within derivative contracts of “VersionedInitializable”. For this purpose, Aave has defined these functions as well as declared a “REVISION” constant that is publicly accessible.

**Recommendations:**
We advise that the function signature of “getRevision” is instead converted to “public”. As constant variables are meant to conform to the UPPER_CASE_FORMAT, a getter function in the form of “getRevision” is more legible and sensible than invoking “REVISION” from off-chain applications.

**Alleviation:**

The AAVE team responded to this Exhibit by stating that the internal styling guideline they conform to utilizes auto-generated getters instead of user-defined ones as they are less error prone and less verbose and as such, this Exhibit is inapplicable.

## Exhibit 3

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inefficient Greater-Than Comparison w/ Zero | Optimization | Informational | token/AaveToken.sol: L135 token/LendToAaveMigrator.sol: L52, L62 |

**[INFORMATIONAL] Description:**

The lines above conduct a greater-than “>” comparison between unsigned integers and the value literal “0”.

**Recommendations:**

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality “!=” reducing the gas cost of the functions.

Additionally, L62 of “migrateFromLEND” in “LendToAaveMigrator” could instead internally call the function “migrationStarted” which would have to be converted to “public”. This would ensure consistency in the checks and enable additional checks to be imposed on “migrationStarted” in the future if necessary.

If no subsequent development is expected, the last point can be safely ignored as it would lead to a miniscule increase in the gas cost of “migrateFromLEND”.

**Alleviation:**

The AAVE team evaluated this Exhibit and proceeded with applying the greater-than to inequality optimization on the aforementioned lines and chose to avoid declaring “migrationStarted” as “public” and retaining L62 as is.

## Exhibit 4

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unconventional Naming of “public” Variables | Coding Style | Informational | token/LendToAaveMigrator.sol: L19, L22 |

**[INFORMATIONAL] Description:**

The variable of L19 (“LEND_AAVE_RATIO”) properly conforms to the naming convention of “immutable” and “constant” variables yet is declared “public”. The variable of L22 does not conform to the naming convention of publicly accessible variables as it is prefixed with an underscore.

**Recommendations:**

For the former, we advise that it is instead set to “private” or “internal” and a dedicated getter function is set that allows off-chain applications to retrieve the ratio. The Solidity compiler automatically generates getter functions for “public” variables and as such, the above statements are equivalent to the current code in terms of gas impact.

For the latter, we advise that the underscore is simply removed from the variable declaration.

**Alleviation:**

The AAVE team proceeded with not changing the aforementioned naming conventions of the variables as the former, L19, is answered by the “Alleviation” chapter of Exhibit 2 and the latter once again is a result of AAVE’s internal styling guide mandating that the “_” prefix is set on all state variables regardless of visibility.

## Exhibit 5

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Truncation of LEND Migration Amount | Mathematical | Minor | token/LendToAaveMigrator.sol: L64 - L67 |

**[MINOR] Description:**

The “migrateFromLEND” function accepts an “amount” input in the form of a “uint256”, transfers the full amount of LEND to the contract and subsequently transfers the AAVE equivalent to the sender by dividing the amount with the “LEND_AAVE_RATIO” variable.

**Recommendations:**

The division with “LEND_AAVE_RATIO” will always truncate if “LEND_AAVE_RATIO” is different than 1 and would lead to the truncated amount being permanently locked within “LendToAaveMigrator”, thus causing the final conversion ratio to be different than the imposed one.

We advise that the modulo of “amount” with “LEND_AAVE_RATIO” is subtracted from itself to calculate the exact amount of LEND that will be migrated and prevent trailing LEND from being locked up in the contract.

**Exhibit 5**

**Alleviation:**

The documentation material provided to us by AAVE indicated that the team was aware of this truncation mechanism and that the solutions they have investigated are not ideal for this type of issue due to their complexity.

The important thing here to note is that trailing LEND as well as trailing AAVE will be locked up in the contract as the LEND token is expected to be fully converted to the AAVE token, meaning truncations will lead to units of both LEND and AAVE remaining out of circulation forever.

As this is undesirable behavior, a novel solution we propose would be to instead store the surplus “LEND” within a variable of the contract that is carried over to the next conversion. This will ensure that when the final transfer of “LEND” to “AAVE” occurs, no trailing “LEND” will remain within the contract.

Additionally, this calculation could also initialize the “remainder” variable at a value that allows the total supply of “LEND” to be fully divisible by the “LEND_AAVE_RATIO” variable.

**Alleviation v2:**

The AAVE team took note of our proposed solution, however due to imminent deadlines they understandably preferred to stick to the original system that has also been properly vetted. We can safely state that the amount that may be locked up in the contract is indeed miniscule and as such should be of no concern.

## Exhibit 6

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Unconventional Naming of “public” Variables | Coding Style | Informational | token/AaveToken.sol: L34, L36, L38, L43, L45, L47 |

**[INFORMATIONAL] Description:**

The variables of L34, L36, L38 and L43 are “public” yet prefixed with an underscore. The “DOMAIN_SEPARATOR” defined in L45 is listed as “public” yet follows the naming convention of “constant” and “immutable” variables and the “PERMIT_TYPEHASH” is declared as “public” correctly following the UPPER_CASE_FORMAT but being illegible for off-chain applications via its compiler-generated getter.

**Recommendations:**

We advise that the first four variable declarations omit the underscore, the “DOMAIN_SEPARATOR” variable is converted to the camelCase format and that a dedicated getter is defined for the “PERMIT_TYPEHASH” variable which should be set to either “internal” or “private”.

If Exhibit 7 is followed, the point about “DOMAIN_SEPARATOR” should be ignored.

Additionally, the variable of L38 could be renamed to “snapshotsLength” instead of “countsSnapshots” to aid in understanding its purpose.

**Alleviation:**

As per the AAVE’s response to Exhibit 2 and Exhibit 4, they chose to retain the naming convention as it is compliant with the team’s internal styling guidelines.

## Exhibit 7

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| EIP712 Adjustment | Coding Style | Informational | token/AaveToken.sol: L45, L65 - L71 |

**[INFORMATIONAL] Description:**

The “DOMAIN_SEPARATOR” variable of EIP712 can be converted to “immutable” and set within the “constructor” by utilizing the EIP1344 which was partially created for EIP712.

**Recommendations:**

The assignment of L65 - L71 should instead be moved to the “constructor” of the contract. As at its current state it relies on the “chainId” parameter, this can be derived using EIP1344 via assembly by using the “chainid()” opcode.

A simple example of utilization would be to declare a “uint256” variable titled “chainId”, declare an assembly block and assign the result of “chainid()” via the “:=” operator to the variable “chainId” and subsequently use it as per the original assignment.

Additionally, the statement of L68 should instead use the current “REVISION” of the contract rather than the string literal “1” to ensure that updates in the codebase are reflected in the EIP712 domain separator.

**Alleviation:**

The AAVE team partially acknowledged this Exhibit by following our recommendation regarding retrieving the “chainId” variable of EIP1344 via assembly.

Our recommendation with regards to setting the “DOMAIN_SEPARATOR” as immutable was avoided as the assignment of the variable relies on the statement “address(this)” which would differ when executed in the constructor, resulting in the address of the logic contract, and when executed in the “initialize” function, resulting in the address of the proxy contract.

It is still feasible to set the variable as “immutable” by passing in the address of the proxy to the constructor of the “AaveToken” logic contract, however we leave this optimization up to the discretion of the AAVE team as they may wish to avoid linking the logic contract with the proxy contract directly.

The rationale behind this optimization is that gas cost will be greatly reduced for “permit” invocations as they would not require to read from state and would instead read the resulting literal on the code itself as that is what the “immutable” trait does.

Additionally, our point with regards to utilizing the “REVISION” variable instead of the string literal “1” on L75 still stands.

**Alleviation v2:**

Similarly to Exhibit 5, the time constraints that the AAVE team had to comply with did not permit major changes in the codebase meaning that the team decided not to alter the deployment procedure of the contracts after weighing the benefit. Additionally, with regards to the “REVISION” variable, the team stated that they “decided to keep (the original implementation) to avoid extra encoding / cast from the uint256 REVISION”.

## Exhibit 8

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Redundant Assignments & "_setupDecimals" Invocation | Ineffectual Code | Informational | token/AaveToken.sol: L72 - L74 |

**[INFORMATIONAL] Description:**

The function "_setupDecimals" is called to set the "decimals" of the ERC20 interface to 18. Internally, the ERC20 interface assigns the literal "18" by default to the value of "decimals". Additionally, the values of "NAME" and "SYMBOL" are set to "_name" and "_symbol" respectively whereas the constructor of "ERC20" is called on L51 which assigns those values as well.

**Recommendations:**

All aforementioned statements can be safely omitted as they are duplicate assignments.

**Alleviation:**

The AAVE team articulated how both contracts are meant to be utilized via a proxy and as such, the points with regards to the redundant assignments are void. The team addressed the "_setupDecimals" invocation by utilizing a constant "DECIMALS" variable instead of the value literal "18".

## Exhibit 9

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inconsistent EIP2612 Implementation | Logical | Medium | token/AaveToken.sol: L47, L91 - L116 |

**[MEDIUM] Description:**

The “typehash” of the permit function as well as its internal statements do not conform to the EIP2612 specification as the “expiration” and “value” variables are swapped in between implementations.

**Recommendations:**

We advise that EIP2612 is conformed to the letter, as the current implementation is misleading. The reason it is misleading is because the function signature matches the specification’s ABI (“address, address, uint256, uint256, uint8, bytes32, bytes32”) yet the “typehash” utilized is different as well as the way the values are utilized.

This could lead to a generic EIP2612 implementation misleading users to input the “value” to be transacted and the “deadline” whilst those two will be utilized in place of one another within the codebase.

As EIPs are meant to streamline the way smart contracts interface with off chain applications on the Solidity ecosystem, we advise that the lines of this Exhibit are amended accordingly to conform to EIP2612.

**Alleviation:**

The AAVE team fully addressed this Exhibit in a dedicated merge-request as it necessitated changes throughout the codebase as well as the test suites.

## Exhibit 10

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Redundant “SafeMath” Utilization | Optimization | Informational | token/AaveToken.sol: L114, L135, L136, L139, L157, L161 token/LendToAaveMigrator.sol: L15, L66 |

**[INFORMATIONAL] Description:**

The “SafeMath” library has been defined to enforce bound checking for any additions and / or subtractions that occur using its exposed methods such as “add” and “sub”. All lines mentioned by this Exhibit either have guaranteed bounds by surrounding code or are logically safe.

**Recommendations:**

The safe addition conducted on L114 for the “currentValidNonce” of an address will never reach the limit of “uint256” as it is sequentially increasing by one and the number of “permit” transactions necessary to reach it are practically infinite.

The lines 135, 136 and 139 of the “_writeSnapshot” function refer to the number of snapshots stored in the “mapping(uint256 => Snapshot)” variable. As this by itself has an inherent limit of “uint256”, safe additions are unnecessary as the number of “Snapshot”s is practically infinite.

Additionally, the safe subtractions that occur on L135 and L136 are done so by subtracting the literal “1” and are preceded by a bound checking ensuring the variable being subtracted from, “ownerCountOfSnapshots”, is above zero.

The safe subtractions and additions done on L157 and L161 of "_beforeTokenTransfer" are able to indeed underflow or overflow, however this type of underflow and / or overflow is captured by the actual "transfer" operations occurring with "ERC20" itself so they can be safely omitted as the "ERC20" transfer will revert causing all previous operations to be reverted as well.

Finally, the safe division done in L66 of "LendToAaveMigrator" is redundant as it simply checks that the right hand of the division is non-zero, a trait guaranteed by what "LEND_AAVE_RATIO" is meant to represent. This makes the import of "SafeMath" entirely redundant in "LendToAaveMigrator".

**Alleviation:**

The AAVE team acknowledged the statements of this Exhibit, however they chose to not apply the optimizations as they desire to keep consistency in the way calculations are carried out throughout their project as well as reduce any potential reviewer's cognitive effort in evaluating the code the statements are surrounded by.

## Exhibit 11

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Declaration Optimization | Optimization | Informational | token/AaveToken.sol: L135, L136, L138 |

**[INFORMATIONAL] Description:**

The variable “block.number” is utilized twice and casted to a “uint128” both times. Additionally, the variable “_snapshots[owner]” is accessed multiple times.

**Recommendations:**

The statement “uint128(block.number)” can be assigned to an in-memory variable to reduce the gas cost of the code path of a snapshot length above zero and a different block number of the latest snapshot than the current one.

Additionally, the statement “_snapshots[owner]” can be assigned to a “mapping(uint256 => Snapshot) storage” local variable. This will significantly reduce the gas cost of the function as the initial lookup of the “mapping” by “owner” will be done once instead of multiple times throughout the statements of the function.

**Alleviation:**

The AAVE team heeded the recommendations of this Exhibit to the letter, declaring the local variables suggested by the recommendations chapter at L139 and L142 respectively.

## Exhibit 12

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Purpose of “initializer” Modifier | Optimization | Informational | token/LendToAaveMigrator.sol: L45 - L46 |

**[INFORMATIONAL] Description:**

The “initializer” modifier as provided by the defacto OpenZeppelin library acts as a guard against multiple executions of a function that initializes a contract’s values

**Recommendations:**

As the “initialize” function of “LendToAaveMigrator” is empty, the “initializer” modifier could instead be used on the “constructor” of the contract. This would allow the removal of the “initialize” function as it serves no purpose at its current state.

If it is envisioned that the “initialize” function will contain code in a next iteration, or that the “initialize” function is needed to be present in the ABI by your deployment processes, then feel free to ignore this Exhibit and retain the code as is.

**Alleviation:**

As per the second paragraph of our recommendations, the AAVE team responded by stating they expect to add more initializing code in the “initialize” function for potential next iterations where it would contain logic which is expected.

## Exhibit 13

| TITLE | TYPE | SEVERITY | LOCATION |
| --- | --- | --- | --- |
| Inexistent Access Control | Logical | Informational | utils/DoubleTransferHelper.sol: L14 - L17 |

**[INFORMATIONAL] Description:**

The function “doubleSend” of the “DoubleTransferHelper” contract conducts two transfer operations with the funds owned by the contract.

**Recommendations:**

We advise that proper access control is imposed on this function as at its current state any party is able to invoke it. The reason this finding was labelled as “informational” is because “DoubleTransferHelper” is a test suite dependency and as such we expect it to not be utilized in a production context.

**Alleviation:**

The AAVE team acknowledged this Exhibit and will not apply it as it is solely a test suite dependency for evaluating two snapshots created within the same block.