CERTORA

AAVE Token Verification

1. Summary

This document describes the specification and verification of the AAVE token using the Certora Prover. The work was undertaken from June 28th 2020 through June 30th, 2020. The latest commit that was reviewed and run through the Certora Prover was 8139d24aca40144847671bed37b43293869b5d51.

The scope of the project was to check compliance of the AAVE token (AaveToken.sol) against a standard ERC20 specification, with additional rule writing for extra functionalities in the AAVE token and the Migrator contract (LendToAaveMigrator.sol).

The Certora Prover proved the AAVE token implementation to be correct with respect to the formal rules written. During the verification process, the Certora Prover discovered a number of bugs in the code listed in the table below in section 1.1. All the high severity issues were promptly corrected, and the fixes were verified to satisfy the specifications. Section 2 formally defines high level specifications of the protocol.

1.1. Issues found

Bug	Affected code	Description	Severity	Fix
Nonces don't increase	permit() function in token contract	A call to permit() checks if the nonces match, but won't increment it in storage. This allows replay attacks	High	Changed safe math call to a postfix ++ operator
Wrong snapshots recorded on self transfer	_beforeTokenTransfer()	If a transfer is issued with the recipient being equal to the sender, the snapshot recorded is greater than the actual balance after the transfer, while it should be unchanged.	High	No snapshot is recorded if sender and recipient are the same.
Migrate loses small amount of LEND for users	migrateFromLEND() function in migrator contract	The amount of LEND is divided by a ratio, so if the LEND amount is not a multiple of the ratio, there is a round-down and precision loss, leading to LEND that is not migrated to AAVE.	Medium - token loss is possible, probably miniscule (for a ratio of 1000, at most 999 wei (10^-18) of LEND can be lost per call to migrate)	No action needed - details about the considerations leading to this decision appear in the README.
Expected nonce is the next nonce	permit() function in token contract	The permit() function checks that the nonce in the message is the nonce in the storage +1	Low	Changed safe math call to a postfix ++ operator
Initialization always possible	initialize() function in token and migrator contracts	If the initializable field becomes true outside of initializer, then it will always be possible to call the initializer	Low - not realizable in current code, but could be a risk during contract upgrades	Initialization library simplified
Approve front-running	approve / transferFrom	A known issue in ERC20 of front-running the approvals	Low - code originated in ERC20 standard implementation	No action needed
Transfer functions can revert	transfer / transferFrom	Overflow in snapshot count	Low - 2^256 overflow when incrementing by 1 is not realistic	No action needed

1.2. Technology overview

The Certora Prover is based on well-studied techniques from the formal verification community. Specifications define a set of rules that call into the contract under analysis and make various assertions about their behavior. These rules, together with the contract under analysis, are compiled to a logical formula called a verification condition, which is then proved or disproved by the solver Z3. If the rule is disproved, the solver also provides a concrete test case demonstrating the violation.

The rules of the specification play a crucial role in the analysis. Without good rules, only very shallow properties can be checked (e.g. that no assertions in the contract itself are violated). To make effective use of Certora Prover, users must write rules that describe the high-level properties they wish to check of their contract. Certora Prover cannot make any guarantees about cases that fall outside the scope of the rules provided to it as input. Thus, in order to understand the results of this analysis, one must carefully understand the specification's rules.

1.3. Disclaimer

The Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of the Certora Prover are scoped to the provided specification, and any cases not covered by the specification are not checked by the Certora Prover.

We hope that this information is useful, but provide no warranty of any kind, express or implied. The contents of this report should not be construed as a complete guarantee that the AAVE token is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2. High Level Specification

The following properties were all verified for the AAVE token.

2.1. ERC20 related

Standard rules:

- Standard ERC20 functions (transfer, transferFrom, approve) are compliant with their standard definitions:
    - check the necessary preconditions, and revert if not met
        - The onTransfer call may revert
    - have the desired effects on the balances and allowances
    - have no undesired changes to balances and allowances
- Impossible to subtract from a balance not owned by the transaction's caller.
- Bounded supply - total amount of AAVE token in circulation is constant at value determined at construction time, and cannot be changed. According to AAVE team, it will be 2,600,000 * 10^18.

2.2. Initialization

- An initialized contract cannot be re-initialized without code change

2.3. Signatures

- Nonces for pre-signed messages must be strictly increasing

2.4. Migration contract

- Every successful migration process must update the balances in both LEND and AAVE.

2.5. Miscellaneous

- No overflows and truncation
    - In snapshot computation - impossible thanks to bounded supply
    - In permit() and snapshot count - unrealistic overflow since values are increased by 1 per transaction.
- Reentrancy
    - In the transfer() flow, the call to onTransfer is after snapshots are updated and before balances are updated. The call is by the Aave governance which is a trusted contract, therefore no reentrancy guards were incorporated. Care should be taken in the design of this contract so that any reentrancy it may introduce is not hazardous and keeps the state consistent.