AAVE

Security Assessment

September 21st, 2020

For :
Aave

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

PK: 970b96f66c3e3c77db81c874b3d02048a9cea893ded728f8365264d129490f0b

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.

What isn’t a CertiK report?

- A statement about the overall bug free or vulnerability free nature of a piece of source code or any modules, technologies or code it interacts with.
- Guarantee or warranty of any sort regarding the intended functionality or security of any or all technology referenced in the report.
- An endorsement or disapproval of any company, team or technology.

Summaries

Project Summary

Project Name	Aave Incentives
Description	Sets of smart contracts to enable stake of Aave-related assets and rewards distribution based on them
Platform	Ethereum, Solidity
Codebase	GitLab Repository

Audit Summary

Delivery Date	Sep. 21, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	Jul. 28th, 2020 - Sep. 21rst 2020

Vulnerability Summary

Total Issues	9
Total Critical	0
Total Major	0
Total Minor	0
Total Informational	9

Findings

ID	Title	Type	Severity
ADM-01	Redundant Usage of `uint128`	Optimization	Informational
ADM-02	Uncommon Naming Convention	Coding Style	Informational
ADM-03	Usage Before Assignment	Optimization & Volatile Code	Informational
ADM-04	Named Return Variable	Optimization	Informational
AIC-01	Redundant Usage of `SafeMath`	Optimization	Informational
STO-01	Uncommon Naming Convention	Coding Style	Informational
STO-02	Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational
STO-03	Implementation Consistency	Volatile Code	Informational
STO-04	Order of `RW` Execution	Volatile Code	Informational

ADM-01: Redundant Usage of `uint128`

Type	Severity	Location
Optimization	Informational	AaveDistributionManager.sol: L35

Description:

The constructor of the contract accepts a `distributionDuration` argument in the form of a `uint128` that is subsequently added to the block.timestamp and stored in an immutable variable.

Recommendation:

As the variable is only utilized in a SafeMath addition, it is more gas-efficient to instead utilize a `uint256` variable as the EVM operates optimally with full-word data types.

Alleviations:

The team opted to consider our references and changed the data type of the `distributionDuration` argument to a `uint256`.

ADM-02: Uncommon Naming Convention

Type	Severity	Location
Coding Style	Informational	AaveDistributionManager.sol: L29

Description:

The linked variable is prefixed with an underscore (_) yet is declared as public.

Recommendation:

We advise that the underscore is omitted per the Solidity style guide.

Alleviations:

The team opted to consider our references and changed the variable name, closely followed the Solidity style guide.

ADM-03: Usage Before Assignment

Type	Severity	Location
Optimization & Volatile Code	Informational	AaveDistributionManager.sol: L53-L59

Description:

In the linked code segment, the `emissionPerSecond` member of the `assetConfig` struct is being assigned to after it has been utilized in the function of `_updateAssetStateInternal` that subsequently invokes `_getAssetIndex`.This can cause `_getAssetIndex` to yield incorrect results depending on whether `emissionPerSecond` should reflect the new value being assigned or the previous one.

Recommendation:

We advise the team to revise the code segment.

Alleviations:

No alleviations.

ADM-04: Named Return Variable

Type	Severity	Location
Optimization	Informational	AaveDistributionManager.sol: L112-L134, L142-L160, L168-L189

Description:

The variable `accruedRewards` is declared and returned directly.

Recommendation:

The variable `accruedRewards` could be directly named in the return type of the function

Alleviations:

No alleviations.

AIC-01: Redundant Usage of `SafeMath`

Type	Severity	Location
Optimization	Informational	AaveIncentivesController.sol: L127

Description:

The preceding line from the linked sub invocation guarantees that the subtraction will never underflow thus rendering the internal check of sub redundant.

Recommendation:

We advise that a raw subtraction is utilized here instead.

Alleviations:

The team opted to consider our references and utilized a raw mathematical subtraction.

STO-01: Uncommon Naming Convention

Type	Severity	Location
Coding Style	Informational	StakedToken.sol: L34-L35

Description:

The linked variable is prefixed with an underscore (_) yet is declared as public.

Recommendation:

We advise that the underscore is omitted per the Solidity style guide.

Alleviations:

The team opted to consider our references and changed the variable name, closely followed the Solidity style guide.

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

Type	Severity	Location
Optimization	Informational	StakedToken.sol: L75, L103

Description:

The linked conditionals conduct a greater-than (>) comparison between an unsigned integer and the value of zero.

Recommendation:

As unsigned integers are restricted to the non-negative range, it is possible to convert these comparisons to inequality ones optimizing their gas cost.

Alleviations:

The team opted to consider our references and converted the comparisons to inequality ones.

STO-03: Implementation Consistency

Type	Severity	Location
Volatile Code	Informational	StakedToken.sol: L151-L153

Description:

In other segments of the codebase, when a value greater than the balance of a user is provided as input a comparison is conducted to affect the minimum between the balance and the input. In this instance, invalid amounts would instead throw due to the `require` check imposed.

Recommendation:

We advise that graceful handling is introduced to prevent code halts.

Alleviations:

The team opted to consider our references, removed the `require` statement and used the `sub` invocation of SafeMath, passing an error message as an argument in case the subtraction failed.

STO-04: Order of `RW` Execution

Type	Severity	Location
Volatile Code	Informational	StakedToken.sol: L182-L184

Description:

The mapping lookup `_stakersCooldowns[from]` is stored to an in-memory variable that is subsequently accessed after the `_getNextCooldownTimestamp` invocation that affects it, potentially preventing it from being reset by the linked `if` clause if it was originally zero and subsequently set to a non-zero value.

Recommendation:

We advise that the order the statements are executed is evaluated.

Alleviations:

The team opted to consider our references and stored the `_stakersCooldowns[to]` to an in-memory variable.