xAAVE

Security Assessment

November 11th, 2020

For :
xAAVE

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

Georgios Delkos @ CertiK
georgios.delkos@certik.io

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	xAAVE
Description	A wrapper token of the AAVE protocol enabling staking and utilizing Kyber for investments.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. d561c6c456a88b388c8d47cfd5a99041d71c424e

Audit Summary

Delivery Date	November 11th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	November 9th, 2020 - November 11th, 2020

Vulnerability Summary

Total Issues	16
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	3
Total Informational	13

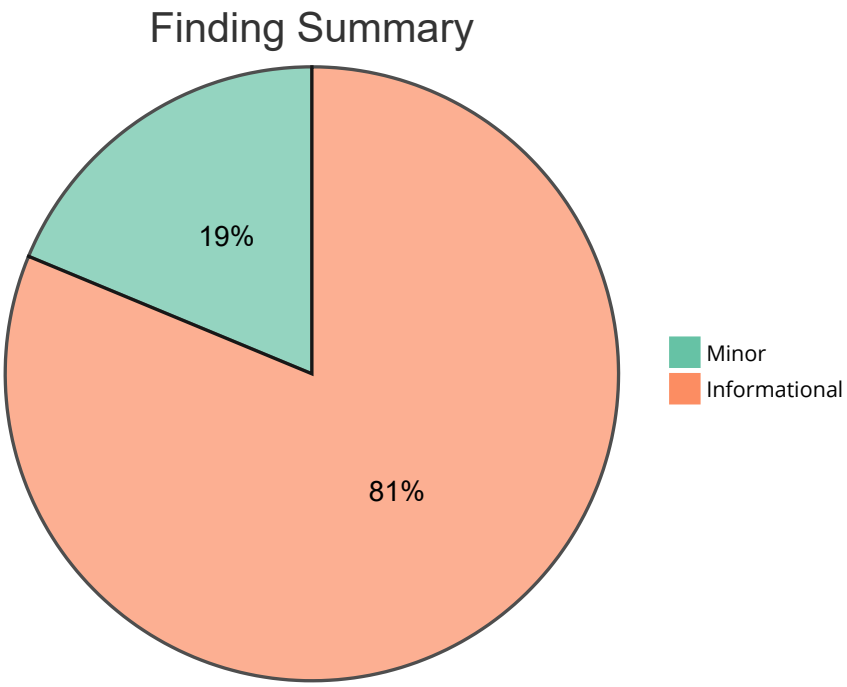
Executive Summary

We were contracted by the xToken team to conduct an audit of their xAAVE x-token passive staking strategy implementation. Our audit was unable to pinpoint any major vulnerabilities; we were able to identify several optimizations that could be applied to the codebase as well as certain security-wise noteworthy unintended-for functionalities that were properly conveyed to the xToken team and took note of.

Files In Scope

ID	Contract	Location
ISA	IStakedAave.sol	contracts/interface/IStakedAave.sol
IKN	IKyberNetworkProxy.sol	contracts/interface/IKyberNetworkProxy.sol
IAP	IAaveProtoGovernance.sol	contracts/interface/IAaveProtoGovernance.sol
PAU	Pausable.sol	contracts/helpers/Pausable.sol
AAV	xAAVE.sol	contracts/xAAVE.sol
AAE	xAAVEProxy.sol	contracts/proxies/xAAVEProxy.sol

Findings



ID	Title	Type	Severity	Resolved
PAU-01	Redundant Variable Initialization	Coding Style	Informational	✓
PAU-02	User-Defined Getters	Gas Optimization	Informational	✓
PAU-03	Functions to Setter	Coding Style	Informational	✓
AAE-01	Unlocked Compiler Version	Language Specific	Informational	✓
AAE-02	Non-Standard Proxy Pattern	Logical Issue	Minor	✓
AAV-01	Visibility Specifiers Missing	Language Specific	Informational	✓
AAV-02	Usage of String Literal	Coding Style	Informational	✓
AAV-03	Redundant Usage of SafeMath	Gas Optimization	Informational	✓
AAV-04	Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call	Volatile Code	Minor	✓
AAV-05	Statement Duplication	Gas Optimization	Informational	✓
AAV-06	Inconsistent Minting / Redemption Ratio	Mathematical Operations	Informational	✓
AAV-07	Imbalanced Allocation w/ Cooldown	Logical Issue	Informational	✓
AAV-08	Function to `modifier`	Coding Style	Informational	✓
AAV-09	Comment Implementation Inconsistency	Logical Issue	Minor	✓
AAV-10	Potential for Lock of Funds	Logical Issue	Informational	✓
AAV-11	Unlocked Compiler Version	Language Specific	Informational	✓

PAU-01: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Pausable.sol L16-L21

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 xToken team removed the redundant `constructor` from the contract thus dealing with this exhibit.

PAU-02: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	Pausable.sol L14, L23-L28

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The xToken team removed the user-defined getter and instead prefixed the linked variable with the keyword `public` to properly utilize the compiler-generated getter function.

PAU-03: Functions to Setter

Type	Severity	Location
Coding Style	Informational	Pausable.sol L54-L76

Description:

The linked function implementations assign a `false` or `true` value to the `_paused` variable depending on which function is called.

Recommendation:

We advise that these functions are instead combined to a single setter function to reduce the bytecode of the contract.

Alleviation:

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

AAE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	xAAVEProxy.sol L21, L28

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The compiler version, as with the other contracts, was properly locked to version `0.6.2`.

AAE-02: Non-Standard Proxy Pattern

Type	Severity	Location
Logical Issue	Minor	xAAVEProxy.sol L208-L236

Description:

The linked `fallback()` implementation contains a non-standard `assembly` block for executing an upgrade-able proxy call.

Recommendation:

We advise that a pattern such as that of OpenZeppelin is followed instead, as the implementation slightly differs i.e. the `switch` case is different and suboptimal.

Alleviation:

The team replaced their previous code with the up-to-date proxy implementation of OZ thus nullifying this exhibit.

AAV-01: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	xAAVE.sol L51-L55, L71, L73, L75

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

Visibility specifiers were properly set accordingly for all variables linked by this exhibit.

AAV-02: Usage of String Literal

Type	Severity	Location
Coding Style	Informational	xAAVE.sol L98

Description:

The linked `__ERC20_init` function call contains a hard-coded value for the name of the token albeit an input variable for its symbol.

Recommendation:

We advise that either both values are hard-coded or both values are passed as variables to ensure consistesncy in the way the function is utilized.

Alleviation:

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

AAV-03: Redundant Usage of SafeMath

Type	Severity	Location
Gas Optimization	Informational	xAAVE.sol L264, L269, L274

Description:

The linked SafeMath statements are guaranteed to never fail their imposed `require` checks, conducting them redundantly so.

Recommendation:

We advise that usage of SafeMath here is avoided to optimize gas cost. While we have noted a few statements that can omit the usage of SafeMath, more statements exist in the codebase that should be taken into consideration if optimization is highly desired.

Alleviation:

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

AAV-04: Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call

Type	Severity	Location
Volatile Code	Minor	xAAVE.sol L159, L208, L504, L550

Description:

The linked `transfer()`/`transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation:

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations. The same paradigm should be applied to the `approve` function invocations as well by using `safeApprove()`.

Alleviation:

The `SafeERC20` library was properly integrated into the contract ensuring that all external ERC20 calls are done safely with regards to their execution.

AAV-05: Statement Duplication

Type	Severity	Location
Gas Optimization	Informational	xAAVE.sol L115-L181

Description:

The functions `mint` and `mintWithToken` contain the exact same workflow apart from the way the AAVE funds are procured, the former using ETH converted to AAVE via Kyber and the latter using AAVE directly.

Recommendation:

We advise that their core minting logic is instead relocated to a single common `internal` function to reduce the bytecode of the contract and aid in its maintainability.

Alleviation:

Both implementations were properly joined into a single `_mintInternal` function implementation, reducing the amount of bytecode generated by the contract and increasing code maintainability.

AAV-06: Inconsistent Minting / Redemption Ratio

Type	Severity	Location
Mathematical Operations	Informational	xAAVE.sol L195, L246

Description:

The mint and burn ratio of AAVE to xAAVE and vice versa is imbalanced as the calculations to assess it utilize multiplications and divisions that are prone to rounding errors. Should a discrepancy occur, an attacker would be able to spam multiple "flash" minting and burning operations to acquire a net profit.

Recommendation:

The net profit acquired by such an operation should be negligible, however it should be taken into consideration and this exhibit serves as a notice.

Alleviation:

We discussed with the xToken team and, as we initially assumed, the gain of this type of attack is negligible in comparison to the gas that would be necessary to achieve it rendering it a safe functionality to remain as is.

AAV-07: Imbalanced Allocation w/ Cooldown

Type	Severity	Location
Logical Issue	Informational	xAAVE.sol L249-L275

Description:

When the cooldown is imposed, `_stake` operations do not actually conduct a staking mechanism meaning that the full balance is ultimately allocated to the buffer balance. Should stakes be re-enabled, a subsequent stake via minting would disproportionately stake AAVE. This should be taken into consideration when re-enabling the protocol.

Recommendation:

This simply serves as a notice.

Alleviation:

While no solution was set in the codebase, the xToken team was properly informed and thus will factor in this type of unintended functioality when utilizing emitted on-chain staking events.

AAV-08: Function to `modifier`

Type	Severity	Location
Coding Style	Informational	xAAVE.sol L331-L341

Description:

The linked `_updateAdminActiveTimestamp` function is invoked at the beginning of certain functions to ensure a variable of the contract is updated when they are executed.

Recommendation:

We advise that it is instead coded as a `modifier` to better illustrate its purpose.

Alleviation:

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

AAV-09: Comment Implementation Inconsistency

Type	Severity	Location
Logical Issue	Minor	xAAVE.sol L470-L472, L487-L489

Description:

The linked comments denote the theoretical limits of the various types of fees xAAVE implements while the linked statements contain their supposed implementation. The comments do not accurately reflect the conditions imposed by the implementation however.

Recommendation:

We advise that the comments or the `require` checks are better updated to correctly reflect the range of fees that should be considered valid by the protocol.

Alleviation:

We re-iterated on this issue with the xToken team and concluded that the `require` checks do indeed ensure that the amounts specified conform to the specification laid out in the comments, however we did note that this type of implementation is illegible in comparison to a traditional multiplication and division for calculating the percentage.

AAV-10: Potential for Lock of Funds

Type	Severity	Location
Logical Issue	Informational	xAAVE.sol L562

Description:

The linked `receive` implementation exists to allow Kyber swaps to deposit ether to xAAVE to be utilized when converting tokens, however no checks are imposed for the depositor's address in the `receive` implementation meaning anyone can deposit Ether to the xAAVE contract incorrectly.

Recommendation:

While those Ether will be redeemable via the `withdrawFees` function of the owner, we still advise that a check is imposed to only allow Kyber contracts to deposit Ether to the contract or, if future interaction with other smart contracts is desired, that the `msg.sender` is not equal to `tx.origin` disallowing direct deposits of Ether by EOAs.

Alleviation:

A `require` check was imposed that ensures the depositor of Ether is not an EOA, thus ensuring errant raw Ether transfers won't occur towards the contract.

AAV-11: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	xAAVE.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one. Additionally, this aids in the code review process as certain optimizations, such as the `immutable` specifier, are available from a particular Solidity version onwards.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The compiler version, as with the other contracts, was properly locked to version `0.6.2`.

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.