CertiK Audit Report for GAMA

CertiK Audit Report for

GAMA

Request Date: 2020-09-21

Revision Date: 2020-09-30

Platform Name: EVM

Contents

CertiK Audit Report for GAMA	**1**
Contents	**2**
Disclaimer	**3**
About CertiK	**3**
Executive Summary	**4**
Testing Summary	**5**
Review Notes	**6**
Introduction	6
Documentation	7
Summary	7
Recommendations	7
Findings	**8**
Exhibit 1	**8**
Exhibit 2	**10**
Exhibit 3	**11**
Exhibit 4	**12**
Exhibit 5	**14**
Exhibit 6	**15**
Exhibit 7	**16**
Exhibit 8	**17**
Exhibit 9	**18**
Exhibit 10	**19**

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 GAMA(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 **GAMA** to discover issues and vulnerabilities in the source code of their **Smart Contract** 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

File	Vulnerabilities	Result
Comptroller.sol	1 Minor	Pass
ComptrollerInterface.sol	0	Pass
CarefulMath.sol	0	Pass
ComptrollerStorage.sol	0	Pass
EIP20Interface.sol	0	Pass
EIP20NonStandardInterface.sol	0	Pass
ErrorReporter.sol	0	Pass
Exponential.sol	0	Pass
Gama.sol	0	Pass
GErc20Immutable.sol	0	Pass
GErc20Delegate.sol	0	Pass
GErc20Delegator.sol	0	Pass
GToken.sol	0	Pass
GEther.sol	0	Pass
GTokenInterfaces.sol	0	Pass
GErc20.sol	0	Pass
InterestRateModel.sol	0	Pass
SafeMath.sol	0	Pass
PriceOracle.sol	0	Pass
UniswapPriceOracleV2.sol	0	Pass
WhitePaperInterestRateModel.sol	0	Pass

Smart Contract Audit
This report has been prepared as a product of the Smart Contract Audit request by GAMA.
This audit was conducted to discover issues and vulnerabilities in the source code of GAMA's smart contract.

TYPE	Smart Contract
SOURCE CODE	gama project in gama_code.zip
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Sep 21, 2020
DELIVERY DATE	Sep 30, 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 **GAMA** team to audit the design and implementation of their ERC 20 token smart contract.

The audited source code link is:

- Gama Source Code:

  gama project in gama_code.zip

  Source Code MD5 Checksum

  3a95a723e309e3279e1f1881f86445c1

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.

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and **are something we advise to be enriched to aid in the legibility of the codebase as well as project**. 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 **GAMA** team or reported an issue.

Summary

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however **1 minor vulnerability was identified during our audit that solely concerns the specification**.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

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**.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version Declaration	Language Specific Issue	Informational	CarefulMath.sol, Comptroller.sol, ComptrollerInterface.sol, ComptrollerStorage.sol, EIP20Interface.sol, EIP20NonStandardInterface.sol, ErrorReporter.sol, Exponential.sol, Gama.sol, GErc20.sol, GErc20Delegator.sol, GErc20Delegate.sol, GErc20Immutable.sol, GEther.sol, GToken.sol, GTokenInterfaces.sol, InterestRateModel.sol, PriceOracle.sol, SafeMath.sol, UniswapPriceOracleV2.sol, WhitePaperInterestRateModel.sol

[INFORMATIONAL] Description:

The compiler version utilized throughout the project uses the “^” prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts. Recommend the compiler version should be consistent throughout the codebase.

Recommendations:

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

(GAMA - resolved) The issue is addressed in “gama_code_20200927.zip”.
Source Code MD5 Checksum d75aacb7a8dec09d0742a9303e3ff3c4

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unused Variable	Ineffectual Code	Informational	Comptroller.sol: L545,L546,L607

[INFORMATIONAL] Description:

Some unused function parameters are declared. Remove or comment out the variable name.

Examples:

- Function parameters “liquidator”, “borrower” and “dst”

Recommendations:

Consider removing the unused function parameters.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Proper usage of “pure” and “view”	Coding Style	Informational	Comptroller.sol: L1386

[INFORMATIONAL] Description:

Function state mutability can be restricted to pure, functions can be declared pure in which case they promise not to read from or modify the state.

Examples:

- Function “getGamaAddress”

Recommendations:

Consider declaring this function as pure.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Naming Convention Utilization	Coding Style	Informational	GTokenInterfaces.sol, GToken.sol, GTokenStorage.sol, GErc20Delegator.sol, Comptroller.sol, ComptrollerInterface.sol, CErc20Interface.sol, CDelegatorInterface.sol, Exponential.sol, InterestRateModel.sol, PriceOracle.sol

[INFORMATIONAL] Description:

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

- Constants should be named with all capital letters with underscores separating words UPPER_CASE_WITH_UNDERSCORES
- Functions other than constructors should use mixedCase
- Variables should use mixedCase

refer to https://solidity.readthedocs.io/en/v0.5.17/style-guide.html#naming-conventions

Examples:

- Constants like : gamaClaimThreshold, gamaInitialIndex, closeFactorMinMantissa, closeFactorMaxMantissa, collateralFactorMaxMantissa, liquidationIncentiveMinMantissa, liquidationIncentiveMaxMantissa,doubleScale,halfExpScale,mantissaOne,borrowRateMaxMantissa,reserveFactorMaxMantissa

- Functions like : _setImplementation, _setPendingAdmin, _setComptroller
- Variables like _notEntered,_mintGuardianPaused,_borrowGuardianPaused

Recommendations:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Proper Usage of “public” and “external” type	Coding Style	Informational	Comptroller.sol: L116, L674, L700,L1109,L1258,L1374 Gama.sol: 148,L161,L189 UniswapPriceOracleV2.sol:L 866

[INFORMATIONAL] Description:

“public” functions that are never called by the contract should be declared “external” . When the inputs are arrays, “external” functions are more efficient than “public” functions.

Examples:

- Functions like : enterMarkets, getAccountLiquidity, getHypotheticalAccountLiquidity, refreshGamaSpeeds, claimGama, getAllMarkets, delegate, delegateBySig, getPriorVotes

Recommendations:

Consider using the “external” attribute for functions never called from the contract.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Misuse of a Boolean Constant	Ineffectual Code	Informational	Comptroller.sol: L268,L399,L458,L530,L595,L644 GErc20Delegate.sol: L25,L37

[INFORMATIONAL] Description:

Boolean constants in code have only a few legitimate uses. Other uses (in complex expressions, as conditionals) indicate either an error or, most likely, the persistence of faulty code.

Examples:

```
if (false) {
        maxAssets = maxAssets;
    }
```

Recommendations:

Consider removing the ineffectual code.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Return value not stored	Optimization	Informational	Comptroller.sol: L1014,L1247

[INFORMATIONAL] Description:

The return value of an external call is not stored in a local or state variable.

Examples:

```
function _supportMarket(GToken gToken) external returns (uint) {
        ...
        gToken.isCToken();
        ...
}
```

Recommendations:

Ensure that all the return values of the function calls are used.

Consider adding “require” statement for isCToken:

require(gToken.isCToken(),”This is not a GToken contract!”);

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Boolean Equality	Optimization	Informational	Comptroller.sol: L143,L1057,L1067,L1076,L1085,L1284,L1291,L1332,L1361

[INFORMATIONAL] Description:

Boolean constants can be used directly and do not need to be compared to true or false.

Example:

```
 if (marketToJoin.accountMembership[borrower] == true) {
            // already joined
            return Error.NO_ERROR;
        }
```

Recommendations:

Consider changing it as following:

```
 if (marketToJoin.accountMembership[borrower]) {...
...}
```

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Missing imports	Optimization	Minor	Comptroller.sol: L9

[Minor] Description:

Missing some imports or the imported file is not existing. This could lead to compiling errors.

```
import "./Supervisor.sol";
import "./Gama.sol";
```

Recommendations:

The Supervisor.sol is not existing in the project, consider removing the import and the "_become" function.
(GAMA - resolved) The issue is addressed in "gama_code_20200927.zip".
Source Code MD5 Checksum d75aacb7a8dec09d0742a9303e3ff3c4

Consider adding the correct imports for "Gama.sol".
(GAMA - confirmed) The issue is addressed in "gama_code_20200929.zip".
Source Code MD5 Checksum 67cb16627eaaf0ea0caee7ec3767d65b

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Missing checks of array length	Optimization	Informational	UniswapPriceOracleV2.sol: L211, L694

[INFORMATIONAL] Description:

Many arrays such as “gTokens_”, “underlyings_” and etc... are used to initialize the “TokenConfig” array. It’s better to ensure the length of all these arrays are exactly the same.

Recommendations:

Consider adding some checks as following:

require(gTokens_.length == underlyings_.length, "array length not same");
require(underlyings_.length == symbolHashs_.length, "array length not same");
…