CERTIK

CertiK Audit Report for ForTube Bank

Contents

Contents 1

Disclaimer 3

About CertiK 3

Executive Summary 4

Testing Summary 5

Review Notes 6

- Introduction 6
- Documentation 6
- Summary 7
- Recommendations 7

Findings 8

Exhibit 1 8

Exhibit 2 9

Exhibit 3 11

Exhibit 4 12

Exhibit 5 13

Exhibit 6 14

Exhibit 7 15

Exhibit 8 16

Exhibit 9 17

Exhibit 10 18

Exhibit 11 19

Exhibit 12 20

Exhibit 13 21

Exhibit 14 23

Exhibit 15 24

Exhibit 16	25
Exhibit 17	26
Exhibit 18	27
Exhibit 19	28
Exhibit 20	29
Exhibit 21	30
Exhibit 22	31
Exhibit 23	32
Exhibit 24	33
Exhibit 25	34
Exhibit 26	35
Exhibit 27	36
Exhibit 28	37
Exhibit 29	38
Exhibit 30	39
Exhibit 31	40
Exhibit 32	41
Exhibit 33	43
Exhibit 34	44
Exhibit 35	45
Exhibit 36	46
Exhibit 37	47

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 ForTube (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 **ForTube** to discover issues and vulnerabilities in the source code of their **ForTube's Bank project Smart Contracts** 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 ForTube.

This audit was conducted to discover issues and vulnerabilities in the source code of ForTube's Bank project Smart Contracts.

TYPE	Smart Contracts
SOURCE CODE	https://github.com/thefortube/bank/tree/bank/contracts
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Jul 8, 2020
DELIVERY DATE	Aug 1, 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 ForTube team to audit the design and implementations of their Bank project Smart Contracts. The audited source code linkis:

- https://github.com/thefortube/bank/commit/15c978700fc550e6616c791f92eb5d354e605a9b

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.

Documentation

The sources of truth regarding the operation of the codebase were extensive and well documented. We would advise the team to fully translate the in-line comments to aid our understanding of the specification implementation and the functionality of the code.

Summary

The codebase of the project was identified to be carefully designed and detailed, as well as properly documented.

While most of the issues pinpointed were of negligible importance and mostly referred to coding standards and inefficiencies, **any minor (or above) flaws** that were identified, **should be remediated as soon as possible to ensure** the contracts of ForTube's team are of **the highest standard and quality.**

Recommendations

Concerning the codebase, the main recommendation we can make is **to consider restructuring the custom libraries to make more optimal use of the Ethereum Virtual Machine**.

Additionally, our original advice, that all our findings were carefully considered and assimilated in the codebase of the project to ensure the highest code standard was achieved, was actualized.

Overall, the codebase of the contracts was refactored to assimilate the majority of the findings of this report, enforced linters and/or coding styles as well as corrected any spelling errors and mistakes that appeared throughout the code **to achieve a high standard of code quality and security.**

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version Declaration & Different versions of Solidity used	Language Specific Issue	Informational	First SLoC of all Contracts

[INFORMATIONAL] Description:

The compiler version utilized throughout the project uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts. Also, the compiler version should be consistent throughout the codebase.

Recommendations:

It is a general practise 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.

Alleviation:

The team heeded our advice and locked the version of their contracts at version 0.5.13, ensuring that compiler-related bugs can easily be narrowed down should they occur.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Naming Convention Utilization	Coding Style	Informational	ExponentLib.sol FixidityLib.sol InterestRateModel.sol LogarithmLib.sol PoolPawn.sol PriceOracles.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:

- Contracts should be in CapWords
- Functions and parameters should be in mixedCase
- Constants should be in UPPER_CASE_WITH_UNDERSCORES

In case the naming conventions are not followed, there should be proper documentation to explain the naming and the purpose of the variable.

Examples:

- Variable “fixed_1” in “FixidityLib.sol” file.
- Variable "Index" has unconventional capitalization in “InterestRateModel.sol” file.
- Function “make_payable” is in snake case when camel case is the defacto of ethereum in “PoolPawn.sol” file.
- Variable “_1” in “PoolPawn.sol” file.

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.

Alleviation:

The team neatly renamed the variables, constants, parameters and functions, following closely the general Solidity naming conventions.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Order of Layout Utilization	Coding Style	Informational	PoolPawn.sol PriceOracles.sol

[INFORMATIONAL] Description:

Solidity defines an Order of Layout that should be followed. In general, inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Functions

Recommendations:

See Exhibit 2.

Alleviation:

Although the team did not follow our advice step-by-step, they reorganized the order of layout in the contracts to match their coding style.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Potential Overflow	Mathematical Operations	Informational	FixidityLib.sol Line 119, 148 PoolPawn.sol Lines 568, 580, 1416-1418

[INFORMATIONAL] Description:

The exponentiation can result in an overflow, as the variable “decimals” is set by admin to an arbitrary "uint" type.

The variable “digits” will overflow if its value is equal to or greater than seventy-seven (77).

Recommendations:

It is possible that the exponentiation will overflow, causing incorrect values to be passed as the value. We advise the utilization of the already-imported “SafeMath” library to conduct the multiplication to ensure no overflow occurs however unlikely.

Alleviation:

The team extensively tested for the cases that can cause an overflow and concluded that this case would not appear through this version of the code.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Unused Function Parameter	Ineffectual Code	Informational	FixidityLib.sol Lines 79, 89, 112

[INFORMATIONAL] Description:

Parameter “fixidity” is declared but never used in the function body.

Recommendations:

Remove or comment out the variable name.

Alleviation:

No alleviation.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Function State Mutability	Coding Style	Informational	FixidityLib.sol Lines 82, 92, 115

[INFORMATIONAL] Description:

Functions can be declared pure in which case they promise not to read from or modify the state.

Recommendations:

Functions “add”, “subtract” and “round_off” state mutability can be restricted to “pure”.

Alleviation:

The team changed the state mutability of the said functions to “pure”.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Potentially Dangerous Operation (Multiplication After Division)	Coding Style	Informational	FixidityLib.sol Lines 45, 59-62 LogarithmLib.sol Lines 52-60 PoolPawn.sol Lines 1436-1440, 1456-1461

[INFORMATIONAL] Description:

Solidity integer division might truncate. As a result, performing a multiply before a division might lead to loss of precision.

Recommendations:

Be overly cautious when performing multiplication on the result of a division.

Alleviation:

The team made great use of the SafeMath library while also avoiding multiplication on the result of a division as much as possible. As a result, the possibility of a bad result, due to truncated digits, is significantly lowered.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Use of the “transfer” Function	Coding Style	Informational	PoolPawn.sol Lines 1887, 1904, 1906

[INFORMATIONAL] Description:

When using the “transfer” function, one should consider some drawbacks that come along with them, such as the hardcoded gas amount that they forwarded to the recipient. If “to” is a contract, the transfer may fail due to gas stipend.

Recommendations:

No recommendation.

Alleviation:

No alleviation.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Proper Usage of “require” and “assert” Functions	Coding Style	Informational	ExponentLib.sol Line 16 FixidityLib.sol Lines 23, 71, 138 LogarithmLib.sol Line 21

[INFORMATIONAL] Description:

The “assert” function should only be used to test for internal errors, and to check invariants.

The “require” function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

In case the “require” function is called, like in line 713 in “PoolPawn.sol” file, a custom error message should be provided.

Recommendations:

Consider using the “require” function, along with a custom error message when the condition fails, instead of the “assert” function on the lines showcased above.

Alleviation:

The team heeded our advice and opted to change from “assert” to “require” function calls.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Greater-Than Comparison with Zero	Mathematical Operations	Informational	Address.sol Line 26 PoolPawn.sol Lines 34, 65, 567, 579, 1016, 1222, 1862 SafeMath.sol Line 30

[INFORMATIONAL] Description:

When comparing variables of unsigned type, it's more efficient gas-wise, while taking into account that any value other than zero is indeed valid.

Recommendations:

Change the condition to check inequality with zero, as it is more efficient regarding unsigned integer variables.

Alleviation:

The team heeded our advice and changed the conditions to check inequality with zero to the highlighted cases.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual Condition	Volatile Code	Informational	FixidityLib.sol Line 147

[INFORMATIONAL] Description:

Unsigned integers cannot be less than zero, so the condition should only check for equality with zero.

Recommendations:

Change the condition to check equality with zero, as it is more efficient regarding unsigned integer variables.

Alleviation:

The team, following our recommendations, alleviated this finding.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Format Long Lines	Optimization & Coding Style	Informational	PoolPawn.sol Lines 1436-1440, 1456-1461

[INFORMATIONAL] Description:

The multiple variable assignments can be grouped into a single assignment by instantiating an instance of the struct in-memory.

Recommendations:

Convert the multiple assignments to a single one with properly formatted key-value assignments on the struct.

Alleviation:

No alleviation.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Potential Overflow/ Underflow	Mathematical Operations	Medium	FixidityLib.sol Lines 78-96 PoolPawn.sol Line 844-845

[Medium] Description:

The “add” and “subtract” functions that implement the Mathematical operations of addition and subtraction in the “FixidityLib.sol” file allows for parameters of type “int256”, while the majority of the Mathematical operations done in the “PoolPawn.sol” file consist of variables of type “uint256”. This can quickly lead to either an overflow or an underflow, due to the nature of the input variables (unsigned and not).

For example, the result of the “add” operation in the line 844 in the “PoolPawn.sol” can overflow if token supplies are large which is highly likely leading to function being uncallable.

Recommendations:

The "add" operation on line 844 should be swapped with a safe alternative to deal with overflows. Additionally, upper and / or lower bounds to the array of tokens being queried could be provided as input parameters.

Alleviation:

The team heeded our advice and adjusted the custom libraries they have implemented for their internal operations, after thoroughly testing for the edge cases highlighted in our description section. Although there is still room for a better resource allocation within the libraries, the use

of SignedSafeMath is a great starting point to narrow down the possibility of an overflow/underflow.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Unconventional Naming	Coding Style	Informational	FixidityLib.sol Line 16 InterestRateModel.sol Line 150 PoolPawn.sol Lines 251, 439

[INFORMATIONAL] Description:

Solidity defines a naming convention that should be followed

Recommendations:

See Exhibit 2.

Alleviation:

The team heeded our advice and changed the unconventional naming of the highlighted variables, constants, parameters and function.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary “Return” Variable	Volatile Code	Informational	PoolPawn.sol Lines 707, 791, 847, 980, 1123, 1284, 1401, 1852, 1910, 1933

[INFORMATIONAL] Description:

Unnecessary/unclear “return” variable. In some cases it is instantiated to zero and explicitly returned.

Recommendations:

Naming the return variable to be descriptive of its role.

Alleviation:

The team heeded our advice and removed the unnecessary code.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Potential Out-Of-Gas	Language Specific Issue	Informational	PoolPawn.sol Lines 442-449, 696-707, 779-792, 829-847

[INFORMATIONAL] Description:

Potential Gas limit exhaustion if too many markets are added.

Recommendations:

No recommendation.

Alleviation:

No alleviation.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Multiple Storage Reads & Writes	Language Specific Issue	Informational	PoolPawn.sol

[INFORMATIONAL] Description:

Assigning a value to a “storage” variable and then changing and re-assigning to the same one can be very taxing to the contract.

Example: Implement "mkts[t]" to store as memory and then re-assign to storage.

Recommendations:

Consider converting to "memory" and then assigning to "storage" to avoid multiple storage reads and writes.

Alleviation:

The team heeded our advice and opted to avoid extensive use of storage reads by either assigning the storage value to a local variable and then using the local variable when needed or using the “memory” keyword when viable, instead of repeatedly reading and writing from storage.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Undocumented Magic Number	Coding Style	Informational	PoolPawn.sol Lines 1319, 1705

[INFORMATIONAL] Description:

Undocumented differentiation of functionality based on magic number "uint(-1)".

Recommendations:

Consider using boolean variables.

Alleviation:

The team added inline comments on the highlighted cases.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Avoid Multiple Assignments	Optimization	Informational	PoolPawn.sol Lines 1044-1048

[INFORMATIONAL] Description:

The function “min” is called on two consecutive occasions to assign the proper value to the same variable.

Recommendations:

Consider nesting "min" calls to avoid 2 assignments.

Alleviation:

The team heeded our advice and neatly bypassed this finding.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
No Transfer “Route”	Volatile Code	Informational	PoolPawn.sol Lines 1880-1909

[INFORMATIONAL] Description:

Code path of "owner != spender", "token == address(0)" and ("msgValue == 0" or "msgValue <= amount") will lead to no transfer being made.

Recommendations:

This case should be further evaluated by the team.

Alleviation:

No alleviation.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Variable Duplicate	Optimization	Informational	PoolPawn.sol Lines 307, 436

[INFORMATIONAL] Description:

Definition of “10**18” in two different formats as constant public variables.

Recommendations:

Evaluate whether the second declaration is necessary.

Alleviation:

The team removed the unnecessary code.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
Simplifying Existing Code	Optimization	Informational	InterestRateModel.sol Lines 89-104

[INFORMATIONAL] Description:

The existing code that spans from lines 79 to 84 can be optimized.

Recommendations:

The lines of code showcased above can be simplified by:

- assigning the result "fixidity.add(cash, borrow)" to "y"
- check if “y” is not equal to zero, then assign “y” the result of "fixidity.divide(borrow, total)"

Alleviation:

The team opted to change their code, following our recommendation as highlighted.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Change of Administrator Procedure	Volatile Code	Informational	InterestRateModel.sol Lines 38-50 PriceOracles.sol Lines 65-77

[INFORMATIONAL] Description:

Change of administrator should follow a propose & accept model to avoid delegating to an unowned address.

Recommendations:

Implementation of a propose & accept model to change admin.

Alleviation:

The team heeded our advice and implemented a model where one can propose an administrator and then the administrator rights can be accepted by the nominee, as per recommendation.

Exhibit 24

TITLE	TYPE	SEVERITY	LOCATION
Reducing Lines of Code	Optimization	Informational	PriceOracles.sol Lines 86-105

[INFORMATIONAL] Description:

Code redundancy in the function body.

Recommendations:

The function body can be branched based on "token == address(0)" to reduce lines of code and redundancy.

Alleviation:

No alleviation.

Exhibit 25

TITLE	TYPE	SEVERITY	LOCATION
New Instantiation of a Struct	Optimization	Informational	PoolPawn.sol Lines 363-373

[INFORMATIONAL] Description:

Current struct filling pattern conducts 7 storage assignments.

Recommendations:

Conduct a single newly instantiated struct storage.

Alleviation:

The team heeded our advice and instantiated a new variable of the custom struct type, filled the struct with the correct value and only then was the mapping reference updated.

Exhibit 26

TITLE	TYPE	SEVERITY	LOCATION
“else” Branch Introduction	Optimization	Informational	PoolPawn.sol Lines 886-891

[INFORMATIONAL] Description:

In this case, only the “if” block is implemented. Perhaps it would be better to implement an “else” block and provide a “require” statement.

Recommendations:

The “else” block should be introduced that asserts "msg.value" is equal to zero in case "t" was supplied.

Alleviation:

The team heeded our advice and implemented the “else” block, as recommended.

Exhibit 27

TITLE	TYPE	SEVERITY	LOCATION
Unclear “require” Condition	Optimization	Informational	PoolPawn.sol Lines 1719-1722

[INFORMATIONAL] Description:

In this case, “msg.value” is required to be greater than or equal “closeBorrowAmount_TargetUnderwaterAsset”.

Recommendations:

Perhaps equality should be required here.

Alleviation:

No alleviation.

Exhibit 28

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary “else-if” Condition	Optimization	Informational	PoolPawn.sol Line 1905

[INFORMATIONAL] Description:

The “else-if” block here is redundant. If code reaches here, “msgValue” will always be equal to zero.

Recommendations:

Replace the “else-if” block with an “else” one.

Alleviation:

The team changed the “else-if” block with an “else” one, as per our recommendations.

Exhibit 29

TITLE	TYPE	SEVERITY	LOCATION
Missing else branch	Function Logics	Major	FixidityLib.sol Lines 111-131

[MAJOR] Description:

The usual definition of round-off function is to approximate the number with the nearest power of 10. The number is rounded up if the first digit of the rounding block is at least 5 and the number is rounded down otherwise. The if “if” clause on line 74 only takes care of the rounding up case and the rounding down case is missing.

Recommendations:

Add an “else” block with the following logics: “v = v - v % t” for the rounding down case.

Alleviation:

The team heeded our advice and implemented the “else” block, following our recommendations.

Exhibit 30

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary case splitting	Function Logics	Informational	FixidityLib.sol Lines 111-131

[INFORMATIONAL] Description:

The function round_off also splits the logics into two separate cases, nonnegative and negative numbers. We believe this is not necessary as the logics for both cases are still the same. Moreover it is more gas efficient and readable. For example

- if “v = -16” and we want to round off the last digit, then “v % 10 = 4” and “4 < 10 / 2”, so the result would be “-16 - 4 = -20”, exactly what we would expect.
- If “v = -238” and we want to round off the last 2 digits, then “v % 100 = 62” and “62 >= 100 / 2”, so the result would be “-238 + 100 - 62 = -200”, exactly what we would expect

Recommendations:

Leave out lines 69 - 73 and change return from “v * sign” to “v”.

Alleviation:

The team heavily investigated this exhibit and concluded that the existing procedure was already covering all the edge cases.

Exhibit 31

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Rounding	Function Logics	Informational	FixidityLib.sol Lines 48-63

[INFORMATIONAL] Description:

The arithmetics in the project is implemented with a precision up to a certain number of decimal digits specified in Fixidity struct. After each operation if the number of decimal digits exceeds Fixidity.digits the result is rounded off. We believe that the function “multiply” does not round the last digit correctly. For example if the number of decimal digits is 2 and we want to multiply 267 and 319 (2,67 and 3,19 in reality). The real product is 8,5173 which would correspond to 852 but the function “multiply” returns 851.

Recommendations:

In the return expression we can change “x2 * y2 / fixidity.fixed_1” to “round_off(fixidity, x2 * y2, fixidity.digits)” to do the rounding.

Alleviation:

The team heeded our advice and changed the return value of the function whilst using their “round_off” function.

Exhibit 32

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Rounding	Function Logics	Minor	FixidityLib.sol Lines 65-76

[MINOR] Description:

The arithmetics in the project is implemented with a precision up to a certain number of decimal digits specified in Fixidity struct. After each operation if the number of decimal digits exceeds Fixidity.digits the result is rounded off. We believe that the function “divide” does not round the result correctly. This exhibit is more severe than the previous one because it can affect more than just the last digit. For example if the number of decimal digits is 2 and we want to divide 1000000 by 300 (10000 by 3 in reality). The real product is 3333,333… which would correspond to 333333 but the function “divide” returns 330000. The problem lies in the error of “reciprocal(fixidity, b)” on line 48 getting exacerbated by multiplication with “a”.

Recommendations:

Let’s try using this code on line 48 instead:

“return multiply(fixidity, a % b, reciprocal(fixidity, b)) + (a / b) * fixidity.fixed_1;”

Here we compute the residue of the division “a % b”, which is at most “b - 1”, so the error after multiplication is more controlled.

Alleviation:

The team heeded our advice and changed the return value of the function, opting for a greater edge case coverage.

Exhibit 33

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Implementation	Function Logics	Minor	FixidityLib.sol Lines 40-46

[MINOR] Description:

The usual definition of the floor function applied on x gives the greatest integer that is less or equal to x. For example the floor function on -0.27 would give -1 but the function “floor” would give 0.

Recommendations:

Let’s try using this code on line 33 instead:

“return v - v % fixidity.fixed_1;”

It is also more gas efficient.

Alleviation:

The team heeded our advice and changed the return value of the function, opting for a greater edge case coverage.

Exhibit 34

TITLE	TYPE	SEVERITY	LOCATION
Integer Overflow	Arithmetics	Major	FixidityLib.sol Lines 48-63

[MAJOR] Description:

The arithmetic expression on line 42 can overflow, for example if the number of decimal digits is 2 the input of the function “multiply” is 10**50 and 10**50, then the result would be unexpectedly - 422425...

Recommendations:

Use SafeMath to prevent integer overflow.

Alleviation:

The team neatly used the SafeMath Library to return the value of the function, opting for a greater edge case coverage.

Exhibit 35

TITLE	TYPE	SEVERITY	LOCATION
Repeated Multiplication of a Constant	Gas Optimization	Informational	InterestRateModel.sol Lines 27, 70, 81

[INFORMATIONAL] Description:

In functions “linearSegment” and “curve2” the constant k, resp. c is power involving two constant “point2” and “e-1e18”. Since “fixidity.power_any” is a gas heavy function and the two aforementioned will be frequently used to calculate loan rate, we believe it would be better if this constant be precomputed outside the function.

Recommendations:

Precompute “point2 ** e-1e18” outside “linearSegment” and “curve2”.

Alleviation:

The team heeded our advice and precomputed the number, as per our recommendations.

Exhibit 36

TITLE	TYPE	SEVERITY	LOCATION
“LoanRate” Calculation not Consistent with Documentation	Inconsistent Documentation	Informational	InterestRateModel.sol Lines 107-129

[INFORMATIONAL] Description:

In README.md the calculation of loan rate is split into 3 cases based on the borrow ratio, whereas in the function “getLoanRate” is split into 5 cases with two additional split points 0.02 and 0.98.

Recommendations:

If this is intended we recommend updating the documentation to be consistent with the codebase.

Alleviation:

No alleviation.

Exhibit 37

TITLE	TYPE	SEVERITY	LOCATION
Multiple Storage Reads	Gas Optimization	Informational	FixidityLib.sol Lines 48-63 ExponentLib.sol Lines 10-34 LogarithmLib.sol Lines 16-65

[INFORMATIONAL] Description:

In functions “multiply”, “power_e”, “log_e” the variables “fixidity.digits” and “fixidity.fixed_1” are repeatedly read from storage, which is very gas inefficient.

Recommendations:

We recommend assigning the values to memory variables first before using, as a call from storage costs 200 gas and a call from memory costs only 3 gas.

Alleviation:

The team heeded our advice and opted to avoid extensive use of storage reads by assigning the storage value to a local variable and then using the local variable when needed, instead of repeatedly reading from storage.