CERTIK

Audit Report

Produced by CertiK

for Animoca Brands

Contents

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 Animoca Brands (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 **Animoca Brands** to discover issues and vulnerabilities in the source code of their **NFT Base Dependency 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 Animoca Brands.

This audit was conducted to discover issues and vulnerabilities in the source code of Animoca Brands' NFT Contracts.

TYPE	Smart Contracts & Token
SOURCE CODE	https://github.com/animocabrands/ethereum-contracts-nft_staking https://github.com/animocabrands/ethereum-contracts-assets_inventory https://github.com/animocabrands/ethereum-contracts-core_library
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	May 05, 2020
LAST REVISION DATE	Aug 20, 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 Animoca Brands team to audit the design and implementations of their NFT smart contracts that is meant to be utilized as a dependency by other projects. The audited source code links are:

- NFT: https://github.com/animocabrands/ethereum-contracts-nft_staking/tree/dbfbdc4357499ae2ca2726124bce7ecc6bc2205b
- Core Library Dependencies: https://github.com/animocabrands/ethereum-contracts-core_library/tree/d141ca5fb91db43f08cbd87f2a2fa454f9a328d5
- Asset Library Dependencies: https://github.com/animocabrands/ethereum-contracts-assets_inventory/tree/691f0af6cfb765aaf133c756f4a66ea143a71dec

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.

After the changes to the NFT libraries were finalized, more updates were pushed to the original repos. The final commit hashes evaluated are as follows:

- NFT: https://github.com/animocabrands/ethereum-contracts-nft_staking/tree/f4f73c2f9c502bea9141770a41768d503255f0b6
- Core Library Dependencies: https://github.com/animocabrands/ethereum-contracts-core_library/tree/64ac2fb6bf3d9ed0500228fd8b188df1e23f81c0
- Asset Library Dependencies: https://github.com/animocabrands/ethereum-contracts-assets_inventory/tree/1ce17a8ec4c2a5e9c9e48c1e54e70ed46edc218a

Documentation

The sources of truth regarding the operation of the contracts were minimal at first and are something we advised to be expanded. The Animoca Brands team quickly took note of our suggestion and expanded the documentation of the NftStaking Contract both in terms of in-line comments as well as a dedicated markdown file that describes its intended purpose and functionality, greatly aiding in our understanding of each contract's functionality.
These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Animoca Brands team or reported an issue.

Summary

The codebase of the project, especially with regards to the NFT Staking concept, attempts to fulfill a use case that is intricate and ambitious and as such, **inefficiencies and flaws in both the design and implementation** of the various contracts were identified and properly documented.

While **most of the issues pinpointed were of negligible importance** and mostly referred to coding standards and inefficiencies, **minor, medium flaws** were identified and swifty remediated as soon as possible to ensure the contracts of the Animoca Brands team are of the highest standard and quality.

We created and maintained a direct communication channel between us and the Animoca Brands team to aid in amending the issues identified in the report. **The final verdict of the audit can be observed in the last section of this document entitled Summary of Round 4.**

Recommendations

With regards to the codebase, the main recommendation we can make is **the expansion of the documentation to address the functionalities of the contracts** from an external perspective rather than an on-code perspective. Additionally, we advise that all our findings are carefully considered and assimilated in the codebase of the project to ensure the highest code standard is achieved.

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**.

All our findings were considered by the Animoca team and were applied where desirable on the codebase of the project, resulting in an exemplary codebase with a good conduct and coding ethics.

Findings (Round 2)

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Incorrect “power-of” Operation	Mathematical Operation	Medium	NftStaking.sol Line 29

[MEDIUM] Description:

The constant variable “MAX_UINT” is meant to represent the maximum value a “uint256” can hold. In the aforementioned line, the expression of that value is stated as “2 ^ 256 - 1”.

Recommendations:

This expression is incorrect as the “^” operand in Solidity is a “XOR” operation (bitwise exclusive-or) rather than a “power-of” operation. Thus, the result of “2 ^ 256 - 1” evaluates to “253”. A “power-of” operation is expressed by two consecutive asterisks “**”, consult Exhibit 6 for the correct statement(s).

Alleviation:

TBA.

Alleviation v2:

The variable was completely removed from the contract as it was deemed unnecessary by the Animoca Brands team.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Tight Packing	Optimization	Informational	NftStaking.sol Lines 43 - 47

[INFORMATIONAL] Description:

All the structs contained in the contract respect Solidity's tight-packing mechanism and are formatted as such to optimize the storage space they utilize. However, the "TokenInfo" struct inefficiently packs its variables.

Recommendations:

At its current state, the variables "address owner" of 160-bits and "uint64 depositTimestamp" are packed in the first 256-bit slot of the struct whereas the variable "uint64 depositCycle" is padded and packed in the second slot of the struct. This is inefficient, as reading the "depositCycle" alone requires unpadding its value.

We advise the change of the "depositTimestamp" variable from a "uint64" to a "uint96" and the "depositCycle" member from a "uint64" to a "uint256" variable as, even when declared as "uint64", the variable will occupy the full 256-bit slot and as a result actually cost more to read as it would need to be unpadded from 256-bits to 64-bits on each read operation.

Alleviation:

TBA.

Alleviation v2:

Instead of restructuring the struct to occupy two slots, the “depositCycle” variable was reduced from a “uint64” to a “uint16”. This does indeed reduce the occupancy of the “struct” to a single 256-bit slot, however the full slot is utilized as the variables contained therein sum up to 240-bits. The final variable should be changed from a “uint16” to a “uint32” to occupy the full slot and reduce the number of un-padding operations required to extract data from the struct.

Alleviation v3:

The struct was altered to instead store the “withdrawCycle” as well, bringing the total utilization of the 256-bit slot to maximum as it was declared as a 16 bit variable as well.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Underscore Naming Convention Utilization	Coding Style	Informational	NftStaking.sol Lines 56 - 58, 60 - 63, 65, 67 - 68

[INFORMATIONAL] Description:

See Exhibit 2.

Recommendations:

See Exhibit 2.

Alleviation:

TBA.

Alleviation v2:

See Exhibit 2 Alleviation.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Visibility Should Be Properly Specified	Coding Style	Informational	NftStaking.sol Lines 54, 69

[INFORMATIONAL] Description:

The declarations contained in lines 54 and 69 do not possess any type of visibility specifier.

Recommendations:

A visibility specifier should be added to denote who has access to the values of these variables in an on-chain sense. If the variables are declared as “internal” or “private”, the underscore naming convention should be kept. Otherwise, the underscore should be removed from the declaration names.

Alleviation:

TBA.

Alleviation v2:

The variable was renamed to “disabled” and properly given a visibility specifier.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Inefficient “greater-than” Comparison	Optimization	Informational	NftStaking.sol Line 79

[INFORMATIONAL] Description:

The “greater-than” comparison conducted inside the “require” statement of Line 79 checks whether the “payoutPeriodLength” is greater than zero.

Recommendations:

As the variable is of type “uint256”, the permitted range of values are greater than or equal to zero, so the “greater-than” comparison can be replaced with a more efficient “not-equal-to” zero since negative values cannot be represented by “payoutPeriodLength”.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Unconventional Function Naming	Coding Style	Informational	NftStaking.sol Line 97, 101

[INFORMATIONAL] Description:

The function declarations contained in the above lines use a mixture of the camel-case and snake-case naming conventions.

Recommendations:

A single naming convention should be utilized and applied across the codebase to ensure consistency in its legibility. In the case of Solidity, we advise that the camel-case is enforced as evident throughout the rest of the codebase.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract's logic.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Logic of Initial Token Distribution	Optimization	Informational	NftStaking.sol Lines 69, 105 - 111, 360, 402, 419, 469, 512, 529

[INFORMATIONAL] Description:

The variable “_initialTokenDistribution” is a “mapping” that maps a “payoutPeriodToClaim” to its token distribution value. In the function “setInitialDistributionPeriod”, a loop is conducted which assigns the “tokensDaily” variable to all mapping positions in between “periodStart” and “periodEnd”.

Recommendations:

This type of operation is highly inefficient for a few reasons. Firstly, since “periodStart” and “periodEnd” are meant to represent a range, the variables themselves could be stored and compared against whatever value was previously used in the mapping to check whether that value is in the range and return the token distribution value.

Secondly, mappings are costly to access as they internally conduct a hashing operation on the input key to find the position of the correct output value. As the output value in the declaration on line 69 is also “uint128”, an additional unpadding calculation also needs to take place. Even if the original scheme is meant to be kept, we advise that the variable is set to a “uint256” as it will cost less to conduct read operations on it.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract's logic.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Inexistence of “reason” String in Require Statements	Optimization	Informational	NftStaking.sol Lines 114, 132

[INFORMATIONAL] Description:

The “require” statements provide no “reason” string for when they fail.

Recommendations:

A corresponding error code should be provided, as done with all other “require” statements, to ease the debugging of smart contracts utilizing “NftStaking” as a base. To note, error code “9” is unused in the sequential list of codes mentioned in the comment block before the contract declaration.

Alleviation:

TBA.

Alleviation v2:

All “require” statements found in the overhauled version contain a corresponding error message, indicating that the team has taken this Exhibit into account.

.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Incorrect ERC1155 Implementation	Ineffectual Code	Minor	NftStaking.sol Lines 138 - 140

[MINOR] Description:

The “supportsInterface” function is meant to support the ERC-1155 Token Receiver implementation as specified in the corresponding EIP-1155.

Recommendations:

In the specification, it is stated that “The implementation MAY differ from the above (example given in EIP)” but “It MUST return the constant value true if 0x01ffc9a7”. This is not true for the aforementioned function as it only returns true for the value “0x4e2312e0”.

Alleviation:

TBA.

Alleviation v2:

Changes were made to the “assets_inventory” repository by Animoca which is a dependency of NFT and through the inheritance chain NftStaking -> ERC1155TokenReceiver -> ERC165 the “supportsInterface” function is correctly implemented according to the specification.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary Duplication of Code	Ineffectual Code	Informational	NftStaking.sol Lines 180 - 186

[INFORMATIONAL] Description:

The functions “addPoolProvider” and “removePoolProvider” assign the constant “bool” “true” or “false” to the “_rewardPoolProviders” mapping and otherwise contain the exact same statements.

Recommendations:

The functions could be combined in a single function with the prefix “set”, f.e. “setPoolProviderValidity” or “setPoolProviderParticipation”, and an input variable of type “bool” denoting whether the “_rewardPoolProviders” mapping should be set to “true” or “false”.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Unsafe Addition of Snapshot Reward	Mathematical Operation	Medium	NftStaking.sol Line 191

[MEDIUM] Description:

Within the function “rewardPoolBalanceIncreased”, which is callable by any reward pool provider, an unsafe addition is conducted on the “tokensToClaim” of a “snapshot” based on user input.

Recommendations:

At its current state, it is possible to nullify the rewards of a snapshot by providing a “uint128 amount” that would cause the “tokensToClaim” member of the “DividendsSnapshot” struct to overflow. We advise that the addition is checked against a potential overflow either manually or by utilizing a safe library like “SafeMath” from OpenZeppelin.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Memory & Storage Management	Optimization	Informational	NftStaking.sol Lines 190, 192, 196, 251, 640, 656

[INFORMATIONAL] Description:

The function "_getOrCreateLatestCycleSnapshot" returns a "DividendsSnapshot" struct stored in "memory" whereas in both functions that it is called (L190 & L640) it ends up being replacing itself after being edited (L192 & L656).

Recommendations:

This is very inefficient as we require the copy of the complete "struct" from "storage" to "memory" and vice versa, even when we affect only a single member of the "struct". By returning a "storage" pointer, the assignments on lines 192 and 656 would be unnecessary as the "storage" itself would be directly affected by the statements in the functions. Additionally, other areas of the codebase can also be adjusted or removed such as L225-227 and L233-L235. On a minor note, a type exists in comment line 189.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract's logic.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Storage Access	Optimization	Informational	NftStaking.sol Lines 200 - 202

[INFORMATIONAL] Description:

The code block between 200-202 accesses the “length” member of the “_dividendsSnapshots” array twice whereas it is also transferred to “memory” on line 206.

Recommendations:

The code block should be moved past the “totalSnapshots” declaration of line 206 and utilize the in-memory variable rather than the in-storage “length” member. Note that an optimization step may already be taken here by the compiler and as such, the gas optimization may not be visible in the transactions conducted.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary Memory Declaration	Optimization	Informational	NftStaking.sol Lines 205, 240, 244

[INFORMATIONAL] Description:

The “initialTokensToClaim” variable is meant to contain the tokens that are meant to be claimed at the snapshot. This is initialized at zero and if a conditional is met, it is set to a non-zero value.

Recommendations:

As the conditional utilized a variable that is already passed to “_addNewSnapshot”, the conditional should be instead moved to “_addNewSnapshot” and the last argument of the function should always be the “tokensToClaim” member of the “snapshot” struct on line 244. This would still fulfill the intended functionality without an unnecessary in-memory declaration. The comment on line 238 (which should be moved to the “_addNewSnapshot” function) also needs to be rephrased as it is currently ambiguous.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary Duplication of Code	Optimization	Informational	NftStaking.sol Lines 271 - 273, 284 - 290

[INFORMATIONAL] Description:

The functions “getCurrentPayoutPeriod” and “_payoutPeriodFromCycleAndPeriodLength” are identical with the sole difference between them being the arguments they consume.

Recommendations:

We advise that “getCurrentPayoutPeriod” internally calls “_payoutPeriodFromCycleAndPeriodLength” with the correct arguments to avoid unnecessary duplication of code and reduce the total bytecode of the contract. Additionally, it appears that the second argument of “_payoutPeriodFromCycleAndPeriodLength” is always the “_payoutPeriodLength” member of the contract and as such could be avoided altogether, meaning that the function “_payoutPeriodFromCycle” between lines 280 and 282 can also be removed.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Inefficient “greater-than” Comparison	Optimization	Informational	NftStaking.sol Line 211

[INFORMATIONAL] Description:

See Exhibit 13.

Recommendations:

See Exhibit 13.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Inefficient “if” Statements	Optimization	Informational	NftStaking.sol Lines 211 - 212, 222, 225, 230

[INFORMATIONAL] Description:

The code block between lines 209 and 245 contains multiple “if” statements that could be simplified by following the previous Exhibits regarding memory management as well as the optimization of the “if” clauses contained in this exhibit.

Recommendations:

The “if” conditional on line 211 evaluates the expression “snapshot.cycleRangeEnd != currentCycle - 1”. If it is true, the body of the “if” clause is executed. In that body, the assignment “snapshot.cycleRangeEnd = currentCycle - 1” immediately follows on line 212. This means that if “snapshot.cycleRangeEnd != currentCycle - 1” evaluated to “true”, it will now evaluate to “false”.

On line 222, the “cycleRangeEnd” member of the “snapshot” struct is changed to an arbitrary value. However, on the “if” clause in line 225 the new value is checked against the previous one and if they are different, the “if” clause’s body executes. This means that within the body of the “if” clause located in line 225 the “snapshot.cycleRangeEnd != currentCycle - 1” expression will evaluate to “true”.

As “snapshot.cycleRangeEnd != currentCycle - 1” is equivalent to “snapshot.cycleRangeEnd + 1 != currentCycle”, the “if” clause of lines 230 - 232 can be moved within the body of the “if” clause of line 225 and the “snapshot.cycleRangeEnd + 1 != currentCycle” check can be skipped.

In general, the code block of the “_getOrCreateLatestCycleSnapshot” should be refactored as it is highly convoluted at its current state and unnecessarily so.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Underscore in Struct Member	Coding Style	Informational	NftStaking.sol Line 312

[INFORMATIONAL] Description:

The struct member "_payoutPeriodLength" of "ClaimDivsParams" contains an underscore as a prefix in its name.

Recommendations:

Struct members are never meant to be private and as such, we advise that the underscore be removed from the name to avoid confusion with the contract's state variable "_payoutPeriodLength".

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract's logic.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Type Declaration of Variable	Ineffectual Code	Minor	NftStaking.sol Lines 307, 356, 359 & More

[INFORMATIONAL] Description:

Throughout the codebase the variables “snapshotIndex” as well as “lastSnapshotIndex” are declared as being of type “int” when representing an array index. Additionally, the function “_findDividendsSnapshot” returns an “int” representation of the index.

Recommendations:

An “overflow” of the index variable can occur in the “hypothetical” scenario that the snapshots are generated in perpetuity, as a “uint” is able to represent numbers up to 2^256 - 1 whereas an “int” is able to represent numbers up to 2^255 - 1. Should such a high number of snapshots be achieved, the contract would lock whereas if “uint”s are used the contract would simply “reset” at 2^256 - 1 entries.

Regardless of the hypothetical scenario, the representation of a non-negative index as a signed integer goes against the best coding practices and as such we suggest the issue to be remediated by converting the return value of “_findDividendsSnapshot” to a “uint” or changing all variables contained in the function to “uint”s.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract's logic. However, our observation indicates that a "SignedSafeMath" library is being utilized for operations that concern the "int" variables used throughout the codebase.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Conditions, Statements and Memory Management	Ineffectual Code	Informational	NftStaking.sol Lines 318 - 441, 444 - 559

[INFORMATIONAL] Description:

The codebase of the functions “estimatePayout” and “claimDividends” are quite complex, hard to understand and unoptimized, rendering it difficult to evaluate their actual functionality.

Recommendations:

There are numerous steps that can be taken to optimize these code blocks. First and foremost, the workaround of the stack limit via the “ClaimDivsParams” struct indicates that an excessive amount of variables is utilized. Since duplicate functionality and statements are contained in the code blocks of “estimatePayout” and “claimDividends”, more functions should be created that are independently called by them and fulfill their functionality with a limited number of input variables that are cleared once their execution ends. This will remove the need for a workaround to the stack limit.

With regards to optimizations, a few highlights will be denoted. The independent “if” clauses of L319-321 + L327-329 and L445-447 + L449-451 should be combined into a single statement as they contain the same function body. Additionally, the latter “if” clauses should be converted to a “require” statement with a corresponding error code rather than simply return, as this may mislead external contracts that interact with “claimDividends” to the function’s correct execution.

The code block of L332-334 is meant to act as a countermeasure against the potential overflow of L351, however an overflow can still occur if a “startPayoutPeriod” that does not overflow when combined with “payoutPeriodsToClaim” is provided and L341 evaluates true, ultimately assigning a value greater than “startPayoutPeriod” to “params.payoutPeriodToClaim”.

Additionally, it alters its input and continues its execution rather than validating it, which goes against both contemporary coding and security practices.

The statements on L346, L369, L401, L403 & L420 (Similarly, L384 & L393) essentially conduct the reverse operation of “_payoutPeriodFromCycle” where “_payoutPeriodFromCycle” should not be invoked in the first place to skip the superfluous computations and a different approach should be utilized. This also applies to L468, L477, L511, L513 & L530.

Unnecessary conversions are contained between “uint” and “int” that would be amended by applying Exhibit 27.

However highly unlikely, line 360 will overflow when processing the last payout period before the “uint” limit divided by “_payoutPeriodLength”.

Many “greater-than” or “less-than” statements can be changed to “not-equal-to”, such as L373, L374, L388 and L406. This also applies to L481, L482, L498 and L516.

On L550, the “state” is assigned from “memory” to “storage”. The only member of the “state” that changes during the course of the function’s execution is the “depositCycle” on L525. As such, it would cost much less gas to simply assign that specific variable on both sides of L550.

A redundant “require” statement is made on L554 prior to the execution of an ERC-20 “transfer”. As the “transfer” function of an ERC-20 internally verifies the balances before transacting them, the check on L554 is redundant and the error code “8” can be provided as a reason to the “require” check on L555.

Overall, if all Exhibits laid out in the report are carried out the size and complexity of the function will be reduced significantly.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Storage Operation	Ineffectual Code	Informational	NftStaking.sol Line 568

[INFORMATIONAL] Description:

At the specified line the statement replaces the “TokenInfo” struct located in “storage” with the one located in “memory” even though only a single member of the in-memory struct is affected

Recommendations:

The affected member should be directly altered in the assignment statement rather than the whole struct to optimize gas cost.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
Incorrect “catch” Block Implementation	Ineffectual Code	Informational	NftStaking.sol Lines 619 - 623

[INFORMATIONAL] Description:

At the specified lines the “catch” block of a low-level EVM error can be observed. However, the exact same statement that caused the error to be thrown is executed.

Recommendations:

The error should be handled differently or not caught at all if bubbling up the error in the call chain is desired. A corresponding error code should also be devised.

Alleviation:

TBA.

Alleviation v2:

The code that this Exhibit concerns remains the same in the latest version of the Animoca Brands contracts.

Alleviation v3:

The “try” block was revamped to instead bubble up the error in case both types of function calls fail.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary Function Invocation	Ineffectual Code	Informational	NftStaking.sol Line 625

[INFORMATIONAL] Description:

On line 625, the function “getCurrentCycle” is executed. However, the same function is executed and stored to an in-memory variable on line 574 after which it remains unaltered.

Recommendations:

The variable “currentCycle” should be used instead when emitting the “Withdrawal” event on line 625.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract’s logic.

Exhibit 24

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Implementation of Concept	Ineffectual Code	Minor	NftStaking.sol Lines 62, 76 - 77, 90 - 92, 571, 637

[INFORMATIONAL] Description:

The "_valueStakeWeights" mapping is utilized to map a "value" denominator to a "weight" denominator and it is initialized during the construction of the contract.

Recommendations:

The core issue with this approach is that the mapping cannot be adjusted after the contract's creation unless derivative contracts implement methods that allow this type of adjustment. This type of functionality should be built-in the "library", as NFT prices, as with any blockchain-based asset, are highly volatile and as such it does not make sense to utilize a static value-to-weight conversion.

Alleviation:

TBA.

Alleviation v2:

No longer applicable due to an overhaul of the contract's logic.

Findings (Round 3)

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Tight Packing	Optimization	Informational	NftStaking.sol Lines 66 - 70, 86 - 90 & more

[INFORMATIONAL] Description:

The “TokenInfo” struct inefficiently packs its variables as there is trailing leftover bit-space. This is observable in other structs such as “NextClaim” and “ComputedClaim” in the codebase.

Recommendations:

In general, it is more optimal to utilize the full bit-space of a tight packed variable rather than truncated versions as this would result in one additional operation of downsizing the variable bits of the trailing pack in a 256-bit segment. Throughout the code, however, the types are used as-is (in the low-bit format) so an additional casting would be required if they were expanded in size. This notice exists as a suggestion to inspect the codebase and make sure that the bit sizes selected fulfill the intended purposes of each variable.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
(OLD) Incorrect “catch” Block Implementation	Ineffectual Code	Informational	NftStaking.sol Lines 282 - 294

[INFORMATIONAL] Description:

The code currently duplicates the exact same statements across two different types of “catch” blocks.

Recommendations:

As the “reason” and “lowLevelData” variables are not utilized, it is possible to replace both “catch” clauses with a single empty clause like so “catch { .. }“ thus reducing the number of statements, logical jumps and subsequently bytecode size.

Alleviation:

The statements were properly updated to instead use a "catch-all" approach and bubble up the error in case the backup system fails.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Contract Address Stored as “address”	Optimization	Informational	NftStaking.sol Lines 106, 107

[INFORMATIONAL] Description:

The variables in the aforementioned lines concern two “immutable” contract addresses, an “IERC20” address and an “IERC1155” address.

Recommendations:

These variables are never utilized as-is, i.e. in their “address” format. As such, instead of casting them to the corresponding contracts on each invocation it is possible to store them as contract-type addresses on construction.

Alleviation:

These variables were declared as being of their corresponding contract type according to our recommendation.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Incorrect “totalPrizePool” Calculation	Contract Freeze	Medium	NftStaking.sol Lines 167 - 186

[MEDIUM] Description:

The “totalPrizePool” variable is incremented every time a new reward period is set via the “setRewardsForPeriods”.

Recommendations:

In case an owner of the contract incorrectly sets a specific period or makes two periods overlap, the previously set reward will not be decremented from the “totalPrizePool”. This means that the “totalPrizePool” will not reflect the actual number of funds necessary to start the contract and could lead to either the contract freezing in case the owner does not have enough funds or the owner sending more funds than redeemable over a normal course of events.

We advise an additional check in the loop that sets the rewards to make sure the storage space being altered is different than zero and if not, to decrement the corresponding value from the “totalPrizePool”.

Alleviation:

A different system was imposed whereby only additions to the current prize pool are feasible instead of adjustments. This in turn leads to the total rewards variable correctly representing the actual total rewards of the various periods, however it does not allow the unsetting of rewards, as per Exhibit 3 of Round 3.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Ability of Owner to Withdraw at Will	Application Design	Informational	NftStaking.sol Lines 214 - 219

[INFORMATIONAL] Description:

The owner is able to withdraw the deposited reward tokens at will via the "withdrawRewardsPool" function.

Recommendations:

This means that, even though the "start" function guarantees enough funds are transferred to the contract to satisfy the reward payouts, the owner would still be able to withdraw them post-start maliciously in case he wants to back-track from providing dividends. This goes against the current paradigm of "funding" the contract on start because the owner can in essence have as many funds as he wishes within the contract.

Alleviation:

An additional check was imposed whereby the owner is only able to withdraw reward tokens when the staking system has been disabled, which should not be the case under normal circumstances.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Multiplication	Mathematical Operations	Informational	NftStaking.sol Line 180

[INFORMATIONAL] Description:

On each “setRewardsForPeriods” call, a multiplication is performed with the “immutable” variable “periodLengthInCycles” and the result of all operations is added to the “totalPrizePool”.

Recommendations:

As the multiplication is done with an “immutable” variable that remains the same across invocations, it is possible to omit it and instead conduct a single multiplication on the result of “totalPrizePool” where it is used on L193. This will reduce the total gas cost necessary for initializing the contract.

Alleviation:

Due to the different system imposed as a solution to Exhibit 4, the multiplication calculation is necessary for each invocation of the function and as such this Exhibit can be considered null.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Contract “start” Access Control	Access Control	Minor	NftStaking.sol Lines 191 - 200

[MINOR] Description:

The “start” function of the contract makes sure that the owner of the contract transmits the necessary funds to conduct the reward payouts and sets the “startTimestamp” variable to “now”.

Recommendations:

There is no guard against calling the “start” function twice so it is possible to “reset” the contract’s “startTimestamp” at any time. As “startTimestamp” is used to get the current cycles in numerous places, if the owner maliciously spams the contract with a “start” transaction and “withdrawRewardsPool” transaction to recoup the amount sent on each “start” transaction it is possible to freeze the contract in numerous sensitive functions, such as “unstakeNft”, as well as affect the logic of the contract in general.

Contracts are meant to give access to their owners on a “need-to” basis rather than an administrative paradigm. This function should be guarded against its execution occurring once by checking whether the “startTimestamp” variable is zero.

Alleviation:

Proper access control was imposed on this function by ensuring that the start timestamp has not been set prior to its execution.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Contract “setRewardsForPeriods” Access Control	Access Control	Minor	NftStaking.sol Lines 167 - 186

[MINOR] Description:

The “setRewardsForPeriods” function can be called at any time on the contract, allowing the owner to overwrite previously set rewards at will.

Recommendations:

The same protection method as Exhibit 7 should be applied, as this function enables the “owner” to in essence “lure” numerous NFT stakes on the platform and subsequently nullify their rewards.

Alleviation:

An additional check was imposed whereby the period the rewards are currently being set for is ensured to be in the future to prevent changing already past / rewarded periods.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Memory Management	Optimization	Informational	NftStaking.sol Lines 608, 609, 614, 615, 626 - 651

[INFORMATIONAL] Description:

The “_updateHistory” function returns a “Snapshot” struct copied via “memory” whilst only a single member of the struct is actually utilized.

Recommendations:

As the function is “internal” and does not exist anywhere else in the codebase, it would be beneficial gas-wise to instead return the “stake” member’s value of a “Snapshot” directly rather than copying the whole struct in memory unless these methods are planned to be utilized by derivative contracts.

Alleviation:

The functions were adjusted to utilize an index instead of the full in-memory struct, optimizing the gas cost of the function.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Unoptimized Code Path	Optimization	Informational	NftStaking.sol Lines 631 - 651

[INFORMATIONAL] Description:

Should “currentCycle” be equal to the snapshot’s “startCycle” and the history provided having at least one snapshot within, the “_updateHistory” function will follow a code-path whereby a full struct is copied from storage to memory, a single member is altered and then the whole struct is once again transmitted from memory to storage, overwriting the previous entry.

Recommendations:

It is possible to account for this case and instead update the “stake” member of the “Snapshot” in-place without reading the whole struct from storage, optimizing the gas cost of the function.

Alleviation:

The location syntax was updated to instead utilize a “storage” pointer.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Exploit Patch	Ineffectual Code	Medium	NftStaking.sol Lines 271 - 274

[MEDIUM] Description:

The codes between L271 and L274 contain a fix to an exploit whereby a user was able to deposit an asset just before the end of a cycle and redeem it at the beginning of the next cycle. This fix merely checks that the cycle difference is equal to or exceeds the number 2.

Recommendations:

The exploit still exists whereby a user deposits an asset just before the end of a cycle and redeems it at the start of the second cycle. In actual time, they would have staked for 1 cycle plus the minimum time that exists before the end and start of a cycle yet have redeemed 2 cycles.

To properly fix this issue, an additional check would be necessary whereby an “eligibility” period is introduced that is calculated by taking into account the exact timestamp the NFT was staked and the current block time a.k.a. “now”. This way, the reward cycle would be calculated by taking into account the cycle the NFT deposit occurred, e.g. Cycle 53, and the number of actual cycles that have passed from the deposit. If the number of cycles is “2.5”, then Cycle 55 would be redeemable rather than Cycle 56 and so on.

Additionally, the comment on L272 contains a spelling mistake (“fukll”).

Alleviation:

This solution would require additional data to be stored on-chain which would significantly affect the logic of the contracts as well as the associated gas cost. As such, after consultation with the Animoca Brands team they chose not to implement this solution.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Loop Iteration	Optimization	Informational	NftStaking.sol Lines 511 - 595

[INFORMATIONAL] Description:

The loop iterates either until the maximum periods to calculate or the current period is met by “claim.periods” and “nextClaim.period” respectively.

Recommendations:

As both variables are incremented by 1 on each iteration, it is possible to simplify the “while” condition to a single logical calculation as well as a single variable increment. The minimum of “currentPeriod” minus “nextClaim.period” and “maxPeriods” is the number of iterations the loop will go through and what “claim.periods” will hold.

As such, one could assign this value to “claim.periods” directly and then instead conduct the “while” loop as long as “nextClaim.period” is not equal to either “currentPeriod” or “nextClaim.period” plus “maxPeriods”, depending on the result of the previous computation. This would reduce the number of computations from two to one and the number of logical comparisons from two to one.

Alleviation:

A calculation was introduced to properly check the number of iterations necessary for the loop and adjust the loop check against one conditional instead of two conditionals to continue its operation.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Mathematical Assumptions	Ineffectual Code	Minor	NftStaking.sol Lines 118, 119, 553 - 559

[MINOR] Description:

The lines of code above pertain to a logical misconception that multiplying by a big enough number, conducting division afterwards with the value we desire and then dividing by the same big enough number would yield higher numerical precision than doing the division directly.

Recommendations:

Solidity rounds numbers down and as such, this type of multiplication and subsequent division is ineffectual. As a side-effect, this also means that the full reward for a cycle will also never be properly divided unless the remainder is kept and handled on a case-by-case basis.

We propose the removal of the superfluous multiplications / divisions and the reconsiderance of how the on-chain dividend mechanism is handled as they are notoriously difficult to implement in an optimized and correct fashion. In the context of rewards, since the administrator is able to withdraw the reward token there is no apparent harm in leaving the remainder in-balance.

Alleviation:

The redundant multiplication and division was omitted from the contract as per our recommendation.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Misleading Event Emittance	Ineffectual Code	Informational	NftStaking.sol Lines 363 - 368

[INFORMATIONAL] Description:

A “RewardsClaimed” event is emitted even if no transfer of funds actually happens in case the rewards to be claimed are equal to zero.

Recommendations:

The event’s emittance should be relocated to the “if” block located between L357 and L361 to ensure that captures of the event properly represent outgoing reward transactions.

Alleviation:

We were informed by the Animoca Brands team that this event emittance is by design and allows the backend services to function appropriately, meaning that the current implementation should be left as is.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Logical Comparison	Optimization	Informational	NftStaking.sol Lines 396, 407

[INFORMATIONAL] Description:

The logical statements of the aforementioned lines check whether the given variables are positive, i.e. above zero, by comparing them using greater-than comparison with zero.

Recommendations:

It costs less gas to perform inequality comparisons rather than greater-than comparisons and as the variable value range has a lower inclusive bound of zero, it is possible to convert the greater-than zero comparison to an inequality comparison with zero, thus saving gas cost.

Alleviation:

The comparison statements were changed to inequality operators to optimize the gas cost of their execution.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Memory Management	Optimization	Informational	NftStaking.sol Lines 404 - 409

[INFORMATIONAL] Description:

Instead of reading the “length” variable directly as exposed by the mapped array, the array is copied in full to memory and then the “length” member is read.

Recommendations:

The “length” member can be directly read from storage without copying the whole array to memory. We suggest it is done so to reduce the gas cost of the function significantly.

Alleviation:

Our advice was heeded and the length member is directly utilized in the code rather than the full array being copied to memory.

Findings (Round 4)

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Redundant Variable Assignment	Optimization	Informational	NftStaking.sol Lines L101 and L103

[INFORMATIONAL] Description:

The contract variables of L101 and L103 assign the value literal 0 as their initial value.

Recommendations:

As Solidity assigns this value by default, these statements are optimized away and as such can be omitted entirely.

Alleviation:

The redundant variable assignments were omitted from the codebase correctly.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Variable Type	Optimization	Informational	NftStaking.sol Lines L193 — L195

[INFORMATIONAL] Description:

In the loop of L193 through to L195 the in-memory variable period is declared as a uint16 which is subsequently used as a uint256 key in the rewardsSchedule mapping.

Recommendations:

As the EVM is more optimized at conducting operations on full-word data types, we advise that period is instead declared as uint256 and that the other variables are casted from uint16 to uint256 instead.

Alleviation:

The counter of the “for” loop was properly set to a “uint256” data type, optimizing the mapping lookup operation.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Immutable Logic	Logical	Minor	NftStaking.sol Lines L194

[INFORMATIONAL] Description:

The current way of setting rewards on L194 does not enable the owner to adjust erroneously set rewards to a lower than the previously set amount.

Recommendations:

The function should either be adjusted to accept a "delta" of the reward as an int data type or the fact that a reward cannot be unset should be documented, the former of which we endorse.

Alleviation:

This trait of the function was instead documented in the function’s accompanying comments and thus, we could consider this Exhibit attended to as the user is properly informed of the consequences of the function.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Struct Instantiation Assignment	Optimization	Informational	NftStaking.sol Lines L466 — L470

[INFORMATIONAL] Description:

The statements of L466 through to L470 can be grouped into a single instantiation and assignment similarly to how L602 does this in the codebase. This will result in zero impact on the generated bytecode and would instead increase the legibility of the codebase.

Recommendations:

Contained within description above.

Alleviation:

The multiple struct member assignments were grouped to a single struct instantiation and assignment.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Redundant Safecast	Optimization	Informational	NftStaking.sol Line L687

[INFORMATIONAL] Description:

As the value of is guaranteed to be positive, the cast of its value to a can occur directly instead of using the safe casting method of uint256 to uint128 as a positive int128 can at most be equal to 2**127 - 1.

Recommendations:

Contained within description above

Alleviation:

The redundant safecast operation was properly removed from the codebase.

Summary of Round 4

The Animoca Brands team has optimized the codebase of the NftStaking implementation to the greatest extent possible applying a high number of optimization techniques as well as compilation rounds on the generated bytecode.

We initially raised our concerns with regards to the way the implementation functions as we anticipated exceedingly high gas costs to result from invocation of the contract's functions, however the Animoca Brands team took note of all our optimization notices and simplified the codebase.

Additionally, the team stated that the current implementation of NftStaking is still experimental in the sense that its design is not fully exploited yet: gamification elements which utilise the current design to a greater extent will be added in future implementations as well as any additional remediations that may become necessary as the project and the feedback it receives evolve.

Overall, the security of the project is on par with what one would expect of a high profile entity and a set of best coding practices and principles can be observed throughout the codebase.