

Secure Code Review

Findings and Recommendations Report Presented to:

Animoca Brands Limited

October 24, 2022

Version: 2.2

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

FOR PUBLIC RELEASE

TABLE OF CONTENTS

TABLE OF CONTENTS	2
LIST OF FIGURES.....	3
LIST OF TABLES	3
EXECUTIVE SUMMARY	4
Overview	4
Key Findings	4
Scope and Rules of Engagement	5
TECHNICAL ANALYSIS & FINDINGS	6
Findings.....	6
KSI-MG-01 – Missing validation check	7
KSI-MG-02 – Design anti-pattern.....	9
KSI-MG-03 – Non-descriptive naming	10
KSI-MG-04 – Variable declaration for constant	11
METHODOLOGY	12
Tools	12
Vulnerability Scoring Systems	13
KUDELSKI SECURITY CONTACTS	14

LIST OF FIGURES

Figure 1: Findings by Severity..... 6

LIST OF TABLES

Table 1: Scope 5

Table 2: Findings Overview..... 6

EXECUTIVE SUMMARY

Overview

Animoca Brands Limited engaged Kudelski Security to perform a secure code assessment of the MotoGP smart contract system on the Flow Web3 platform.

The assessment was conducted remotely by the Kudelski Security Team. The source code review took place from 7/29 – 8/29, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks discovered within the environment during the engagement.
- To provide a professional opinion on code maturity, adequacy, and efficiency of the security measures in place.
- To identify potential issues and include improvement recommendations based on the results of our review and tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The issues found in the code were LOW or INFORMATIONAL findings. This shows that the overall risk profile of the application at the time of this assessment is low.

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce the risk they pose.

- Single administrator accounts have significant capabilities. These functions should be limited by ideally requiring multiple administrators to prevent collusion.

During the test, the following positive observations were noted regarding the scope of the engagement:

- The code is well organized.
- Client contacts were very amenable to conducting joint secure code reviews with the Kudelski Security smart contract auditing team.

Scope and Rules of Engagement

Kudelski performed a Secure Code Review for Animoca Brands Limited on the MotoGP smart contract system on the Flow Web3 platform. MotoGPNFTStorefront.cdc was the target in scope for the engagement.

The source code was supplied through a public repository:

In-scope source code, contracts, or programs	
https://github.com/animoca/motogp-flow-contracts/tree/nftstorefront_mainnet_v1.0.0	Commit: 8cc681d3500b4deddb1494ca9f47ccd552b0ecd2

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, the Kudelski Security team stayed in contact with the development team to share any potential critical issues that could be remediated immediately. We discovered one (1) low severity finding that was resolved and three (3) informational findings that were accepted, but remain open.

The following chart displays the findings by severity.

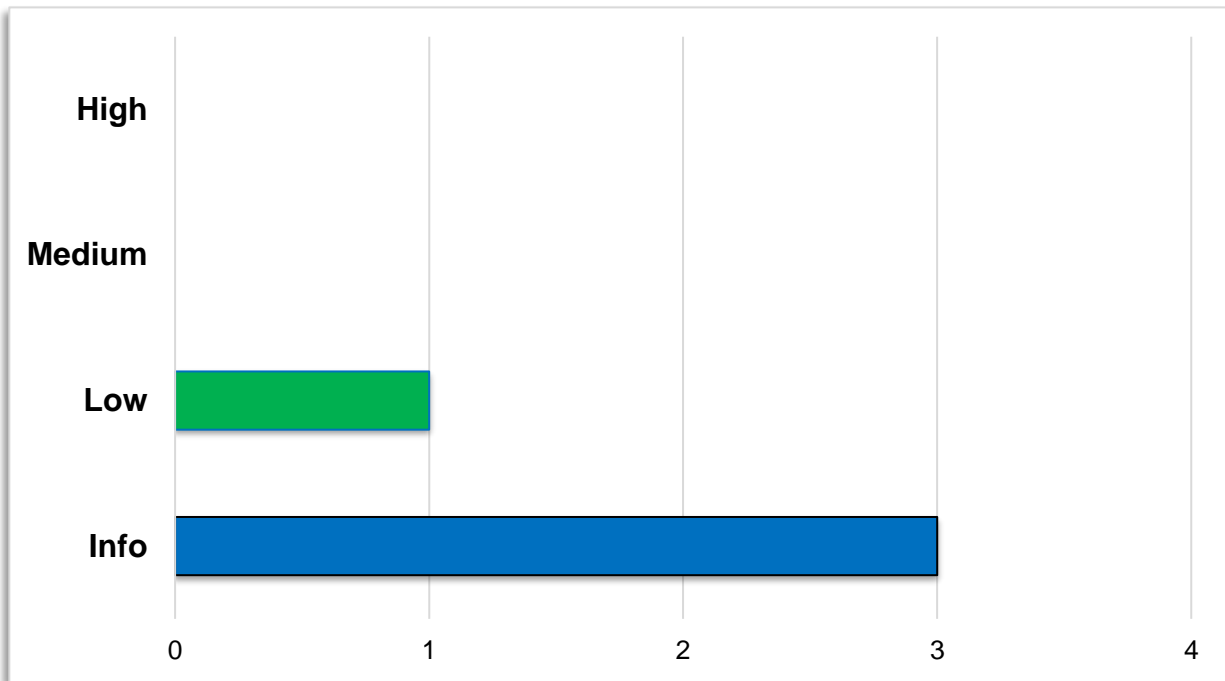


Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	Severity	Description	Resolution Status
01	Low	Missing validation check	Resolved
02	Informational	Design anti-pattern	Open
03	Informational	Non-descriptive naming	Open
04	Informational	Variable declaration for constant	Open

Table 2: Findings Overview

KSI-MG-01 – Missing validation check

Severity	Low
Status	Resolved

Impact	Likelihood	Difficulty
Medium	Low	High

Description

The Kudelski Security team noticed that the *commissionRate* is set without validating that it is within bounds (less than or equal to one).

Impact

Invalid commission rates will invalidate transactions. If *commissionRate* > 1.0, this will yield an underflow error when a new *SaleOffer* is created. Since *sellerPayoutAmount* = *price* * (1.0 - *MotoGPNFTStorefront.commissionRate*).

Evidence

```
MotoGPNFTStorefront.cdc
363         let sellerPayoutAmount = price * (1.0 - MotoGPNFTStorefront.commissionRate)
```

```
MotoGPNFTStorefront.cdc
552     pub fun setCommissionRate(adminRef: &MotoGPAdmin.Admin, commissionRate: UFix64){
553         pre {
554             adminRef != nil : "adminRef is nil"
555         }
556         self.commissionRate = commissionRate
557     }
```

Recommendation

The unsigned type ensures that *commissionRate* will be greater than or equal to zero, but not that it is less than one. Check that the *commissionRate* is less than one within the *setCommissionRate* function.

Remediation

The expression `commissionRate <= 1.0` : "Commission rate is larger than 1.0" inserted on line 555 in the *pre* block of the `setCommissionRate` function ensures that `commissionRate` is within bounds.

Showing 1 changed file with 1 addition and 1 deletion.

```

cadence/contracts/MotoGPNFTStorefront.cdc
@@ -552,7 +552,7 @@ pub contract MotoGPNFTStorefront {
552 552     pub fun setCommissionRate(adminRef: &MotoGPAdmin.Admin, commissionRate: UFix64){
553 553         pre {
554 554             adminRef != nil : "adminRef is nil"
555 -         commissionRate >= 0.0 && commissionRate <= 1.0 : "Commission rate is outside of allowed range"
555 +         commissionRate <= 1.0 : "Commission rate is larger than 1.0"
556 556         }
557 557         self.commissionRate = commissionRate
558 558     }

```

The finding was resolved in commit ([eb2ddb86f1832b435cdf3748102cf4640279c770](https://github.com/AnimocaBrands/Animoca-Brands-Blockchain-Contracts/commit/eb2ddb86f1832b435cdf3748102cf4640279c770)).

KSI-MG-02 – Design anti-pattern

Severity	Informational
Status	Open

Impact	Likelihood	Difficulty
N/a	N/a	N/a

Description

The Kudelski Security team noticed that the struct *SaleOfferDetails* appears to be a script accessible report but contains functions and variable fields.

Impact

This may encourage future developers to add member functions and writable variables, leading to future vulnerabilities within writable fields and functions.

Evidence



```

163         // setToAccepted
164         // Irreversibly set this offer as accepted.
165         //
166         access(contract) fun setToAccepted() {
167             self.accepted = true
168         }

```

Recommendation

The *SaleOfferDetails* struct should contain only constant fields, with all other functionality moved to the *SaleOffer* resource.

References for this finding:

<https://developers.flow.com/cadence/design-patterns#script-accessible-report>

KSI-MG-03 – Non-descriptive naming

Severity	Informational
Status	Open

Impact	Likelihood	Difficulty
N/a	N/a	N/a

Description

The Kudelski Security team noticed that non-descriptive naming such as *details* is used.

Impact

The naming and usage of this struct obscures what would logically be members of the *SaleOffer* resource with descriptive names such as *accepted*, *comissionCut*, and *sellerPayoutCut*. This may lead to confusion and the potential introduction of future vulnerabilities.

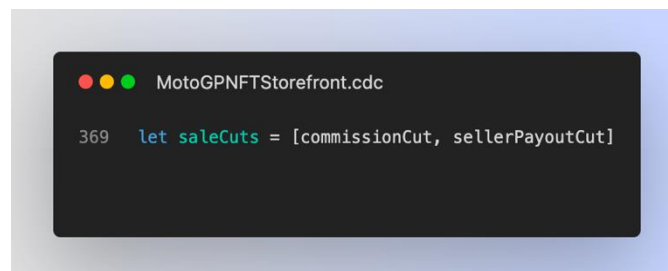
Evidence



```

234 pub resource SaleOffer: SaleOfferPublic {
235     // The simple (non-Capability, non-complex) details of the sale
236     access(self) let details: SaleOfferDetails

```



```

369 let saleCuts = [commissionCut, sellerPayoutCut]

```

Recommendation

In the *SaleOffer* resource, remove the member *details*. Replace it with descriptive names such as *accepted*, *comissionCut*, and *sellerPayoutCut* as members of the *SaleOffer* resource. These correspond to relevant field names in the *details* struct.

References for this finding:

<https://developers.flow.com/cadence/design-patterns#use-descriptive-names-for-fields-paths-functions-and-variables>

KSI-MG-04 – Variable declaration for constant

Severity	Informational
Status	Open

Impact	Likelihood	Difficulty
N/a	N/a	N/a

Description

The Kudelski Security team noticed that the variable *storefrontID* is not modified anywhere in the code and can therefore be declared as a constant.

Impact

Declaring the *storefrontID* field as a variable introduces the possibility that it may be changed inadvertently in the future leading to unexpected results and possible vulnerabilities.

Evidence



```

144     pub struct SaleOfferDetails {
145         // The Storefront that the SaleOffer is stored in.
146         // Note that this resource cannot be moved to a different Storefront,
147         // so this is OK. If we ever make it so that it *can* be moved,
148         // this should be revisited.
149         pub var storefrontID: UInt64

```

Recommendation

Declare *storefrontID* as a constant using the *let* keyword.

References for this finding:

<https://developers.flow.com/cadence/language/constants-and-variables#constants-and-variable-declarations>

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- Authentication
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography
- Security Best Practices

These categories incorporate common Cadence vulnerabilities such as:

- Numerical precision errors
- Capability exposure
- Design anti-patterns

Tools

The following tools were used during this portion of the secure code audit.

- Visual Studio 2022
- Visual Studio Code

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system

Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

High:

The vulnerability has a severe effect on the company and systems or has an effect within one of the primary areas of concern noted by the client

Medium:

It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.

Low:

There is little to no effect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, and institutional knowledge

High:

It is extremely likely that this vulnerability will be discovered and abused

Medium:

It is likely that this vulnerability will be discovered and abused by a skilled attacker

Low:

It is unlikely that this vulnerability will be discovered or abused when discovered.

Difficulty

Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

High:

The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Medium:

The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.

Low:

The vulnerability is easy to exploit or has readily available techniques for exploit

Severity

Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Rez Khan	Blockchain Security Expert	Rez.Khan@KudelskiSecurity.com