Security Assessment for REVV Flow Smart Contracts

Findings and Recommendations Report Presented to:

Animoca Brands Corporation Limited

November 30, 2021

Version: 2.0.0

Presented by:

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

STRICTLY CONFIDENTIAL

TABLE OF CONTENTS

TABLE OF CONTENTS 2
LIST OF FIGURES 2
LIST OF TABLES 3
EXECUTIVE SUMMARY 4
Overview 4
Key Findings 4
Scope and Rules of Engagement 5
TECHNICAL ANALYSIS & FINDINGS 7
Findings 8
Technical analysis 8
Technical Findings 9
General Observations 9
Verification of the REVVVaultAccess contract 10
Verification of the REVV fungible token contract 12
METHODOLOGY 14
Kickoff 14
Ramp-up 14
Review 15
Code Safety 15
Technical Specification Matching 15
Reporting 16
Verify 16
Additional Note 16
The Classification of identified problems and vulnerabilities 17
Critical – vulnerability that will lead to loss of protected assets 17
High - A vulnerability that can lead to loss of protected assets 17
Medium - a vulnerability that hampers the uptime of the system or can lead to other problems 17
Low - Problems that have a security impact but does not directly impact the protected assets 17
Informational 17
KUDELSKI SECURITY CONTACTS 18

LIST OF FIGURES

Figure 1: Findings by Severity 7
Figure 2: Methodology Flow 14

LIST OF TABLES

Table 1: Scope 6
Table 2: Findings Overview 8

EXECUTIVE SUMMARY

Overview

Animoca Brands Corporation Limited engaged Kudelski Security to perform a Security Assessment for REVV Flow Smart Contracts.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on November 11 - November 26, 2021, and focused on the following objectives:

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

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

Key Findings

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 to the risk they pose.

- KS-ANIREVV-01 – Verification of the REVVVaultAccess contract
- KS-ANIREVV-02 – Verification of the REVV fungible token contract

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

- The team was very supportive and open to discuss the design choices made

Based on the formal verification we can conclude that the reviewed code implements the documented functionality. As of the issuance of this report, **all findings** have been **resolved to our satisfaction**.

Scope and Rules of Engagement

Kudelski performed a Security Assessment for REVV Flow Smart Contracts. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/animoca/revv-flow-contracts
with the commit hash 226a373c09aaed8cf5db93af8e169810f730aa56.

Files included in the code review

```
.
├── cadence/
│   ├── contracts/
│   │   ├── REVV.cdc
│   │   └── REVVVaultAccess.cdc
│   ├── scripts/
│   │   ├── get-max-amount-for-account.cdc
│   │   ├── get-proxy-vault-addresses.cdc
│   │   ├── get-revv-balance.cdc
│   │   ├── get-revv-escrow-balance.cdc
│   │   ├── get-total-authorized-amount.cdc
│   │   ├── get-vault-guard-max.cdc
│   │   └── get-vault-guard-paths.cdc
│   └── transactions/
│       ├── borrow-vault-guard.cdc
│       ├── burn-revv.cdc
│       ├── create-vault-guard.cdc
│       ├── create-vault-proxy.cdc
│       ├── failed-access-escrow.cdc
│       ├── failed-assign-to-revv-admin-storage-path.cdc
│       ├── failed-assign-to-revv-balance-public-path.cdc
│       ├── failed-assign-to-revv-max-supply.cdc
│       ├── failed-assign-to-revv-receiver-public-path.cdc
│       ├── failed-assign-to-revv-total-supply.cdc
│       ├── failed-assign-to-revvvaultaccess-adminstoragepath.cdc
│       ├── failed-assign-to-revvvaultaccess-guardtoproxymap.cdc
│       ├── failed-assign-to-revvvaultaccess-proxytoguardmap.cdc
│       ├── failed-assign-to-revvvaultaccess-totalauthorizedamount.cdc
│       ├── failed-assign-to-revvvaultaccess-vaultproxypublicpath.cdc
│       ├── failed-assign-to-revvvaultaccess-vaultproxystoragepath.cdc
│       ├── failed-assign-to-revv-vault-private-path.cdc
│       ├── failed-assign-to-revv-vault-storage-path.cdc
│       ├── failed-bitshift-revv-total-supply.cdc
│       ├── failed-create-revv-admin-outside-contract.cdc
│       ├── failed-create-revvvaultaccess-admin.cdc
│       ├── failed-create-vaultguard-exceeding-max.cdc
│       ├── failed-create-vaultguard-without-admin.cdc
│       ├── failed-deposit-to-escrow-from-outside-revv-contract.cdc
│       ├── failed-emit-mint-event.cdc
│       ├── failed-mint-outside-contract.cdc
│       ├── failed-modify-vault-guard-max.cdc
│       ├── failed-modify-vault-guard-total.cdc
│       ├── failed-vault-withdraw-from-balance-public-path.cdc
│       ├── failed-vault-withdraw-from-provider-public-path.cdc
│       ├── failed-vault-withdraw-from-storage-path.cdc
│       ├── provision-revv-vault.cdc
│       ├── revoke-vault-guard.cdc
│       ├── set-vault-guard-cap-on-vault-proxy.cdc
│       ├── transfer-revv.cdc
│       ├── withdraw-from-revv-mint-vault.cdc
│       └── withdraw-from-vault-proxy.cdc
├── babel.config.json
├── flow.json
├── jest.config.js
├── package.json
├── README.md
└── yarn.lock
```

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Security Assessment for REVV Flow Smart Contracts, we discovered:

- 2 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

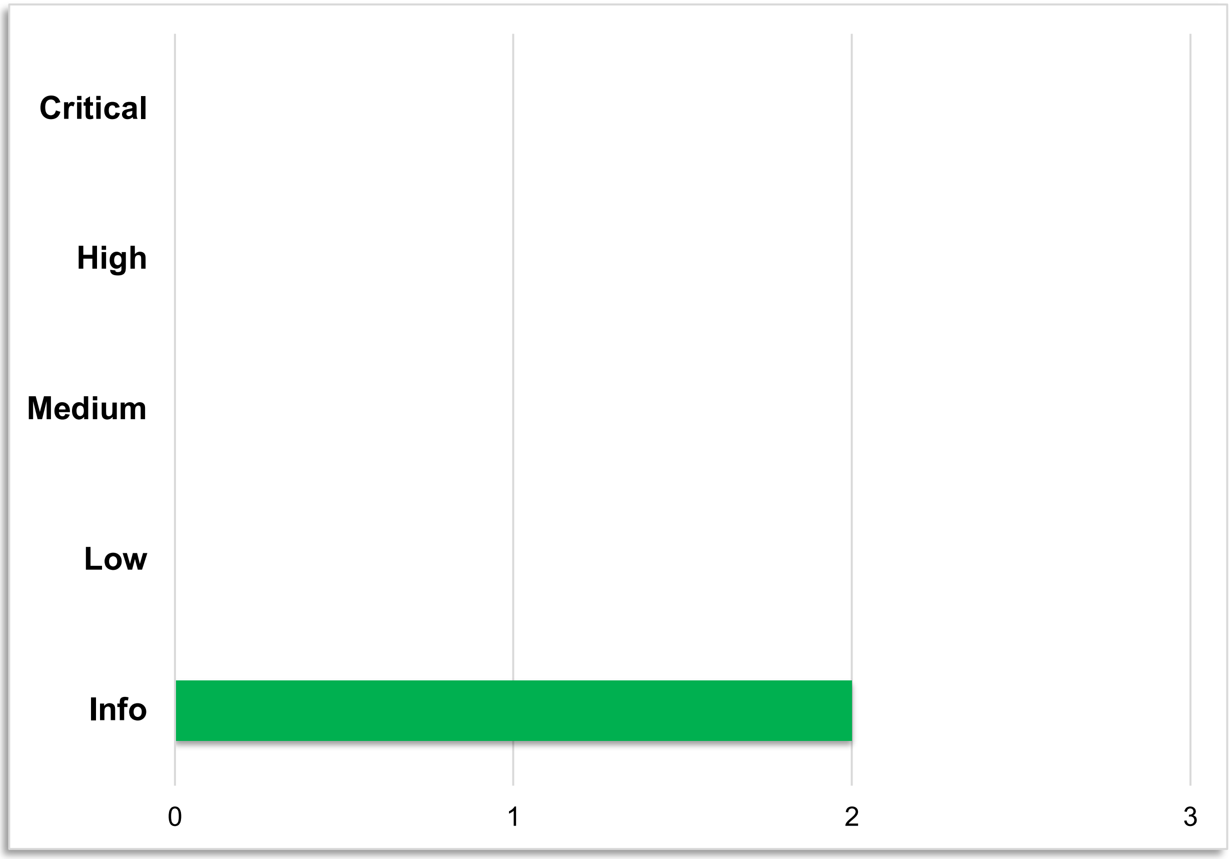

Figure 1: Findings by Severity

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
KS-ANIREVV-01	**Informational**	Verification of the REVVVaultAccess contract
KS-ANIREVV-02	**Informational**	Verification of the REVV fungible token contract

Table 2: Findings Overview

Technical analysis

The validity of the source code was verified to confirm that the intended functionality was implemented correctly and to the extent that the state of the repository allowed.

Based on formal verification we can conclude that the code implements the documented functionality to the extent of the code reviewed.

Technical Findings

General Observations

The reviewed smart contracts implement (1) fungible token functionality and (2) a purpose-built access control contract to allow withdrawal of tokens from the Revv Vault. In particular, the second contract was clearly built with the purpose of token transition, as described by the developers in the launch-meeting.

The code is well built, follows Cadence patterns and is clear to follow. The project is well documented, both in the readme file and the code itself, which further assisted the audit.

The tests provided run the transactions and contracts in great detail.

Comment for the developers: The audit was performed on Mac OS. The reviewer tried deploying and testing the contracts on two Ubuntu builds to no success. We recommend providing further instructions on how to deploy in a Linux environment, or specifying the installations needed (the Cadence documentation is quite short in this regard).

The team was collaborative and responded to the reviewer’s questions quickly.

Verification of the REVVVaultAccess contract

Finding ID: KS-ANIREVV-01
Severity: Informational
Status: [Resolved]

Description

This contract is designed to grant access to accounts to withdraw REVV from the granting accounts vault. Key features to be verified are:

- Feature (1): only the vault contract owner can grant access
- Feature (2): the owner sets a maximum withdraw limit
- Feature (3): the contract owner can equally REVOKE access

Proof of issue (verification)

File name: REVVVaultAccess.cdc
Line number: full file

Feature (1):

- Authorization is granted through the function `createVaultGuard` which stores a reference to the vault through the provider interface (allowing withdrawals).
- The authorized accounts to withdraw REVV from the vault are stored in dictionaries (declared in lines 44 and 49) **WHICH ARE ONLY ACCESSIBLE FROM THE CONTEXT OF THE CONTRACT** by means of the `access(contract)` access control statement.
- Withdraws are executed by calling withdraw function of the `VaultProxy` resource after the owner has granted `vaultGuardCap` capability. Only the admin account can do this.

Feature (2):

- The maximum to be withdrawn by a given guard is set in the constructor of the `VaultGuard` resource. The vault guard is then created by procedure described above.

Feature (3):

- In the `pre`-block the admin reference is checked. This reference is only accessible from the context of the contract (stored in `self.AdminStoragePath`, which should is not public).)** (Question for devs: for extra security, does the `Admin` resource need to be declared public?**)
- The `revokeVaultGuard` function, which checks the admin reference before execution, is in charge of revoking the guard. The specific guard is loaded from the dictionaries and destroyed (the dictionaries can only be modified in the context of the contract).

For the developers to check

Does the `Admin` resource and its creation function need to be declared as public?

Severity and Impact summary

No security issues were detected in the implementation of the contract.

Recommendation

Check the destroy function in case the case `self.balance==0` needs explicit mention.

Verification of the REVV fungible token contract

Finding ID: KS-ANIREVV-02
Severity: **Informational**
Status: **[Resolved]**

Description

The `REVV.cdc` contract implements the `REVV` fungible token for the project. The token is to be given in exchange for the ERC20 REVV tokens as the project migrates to the Flow blockchain. As per the provided documentation, key characteristics of this token are:

- Feature (1) A fixed supply of `REVV` at 3,000,000,000 units to be minted at contract initialization.
- Feature (2) No further minting is allowed.
- Feature (3) `REVV` tokens are not to burned, as this would reduce the supply of tokens. Instead, the `destroy` functionality is implemented so that instead of being burnt the tokens are transferred to a secure escrow.
- Feature (4) The secure escrow where to-be burned tokens end up does not yet have withdraw functionality.

Proof of issue (verification)

File name: REVV.cdc
Line number: full file

Features (1) and (2)

`self.MAX_SUPPLY` for contract is set to the stated amount in line 160, in the initialization call. The tokens are minted by the `self.mint` call on line 212.

The `mint` method is defined from line 143 onwards and is only accessible from within the context of the contract. The pre-execution check enforces that the `amount`+ existing supply `self.totalSupply` is smaller than or equal to `self.MAX_SUPPLY`. If this condition doesn't hold the execution fails.

Feature (3)

The `destroy` method of the REVV token vault is defined in line 103 and is designed for burning vaults containing a balance larger than 0 (to not affect normal token transfer functionality, i.e. so that temporary vaults which allow exchange of tokens can be destroyed as required once their balance is 0.0). In this case the `destroy` call calls the `depositToEscrow` method, only accessible from the context of the contract. This takes the amount to be burned and deposited in the `escrowVault` instance of `Vault`.

Feature (4)

The `escrowVault` instance of `Vault` is only accessible in the context of the contract. An interface is provided to interact with the `escrowVault`, be means of the public function `getEscrowBalance`, which simply checks the number of tokens in this vault. Indirectly users can deposit in the escrow by means of the `destroy()` method, which has been overwritten. No other access to `escrowVault` is possible.

For the developers to check

The overwriting implementation of `destroy()` only contemplates the case when `balance>0.0`. Does the case `balance == 0.0` need explicit implementation?

Severity and Impact summary

No security issues were detected in the implementation of the contract.

Recommendation

Check the destroy function in case the case `self.balance==0` needs explicit mention.

METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

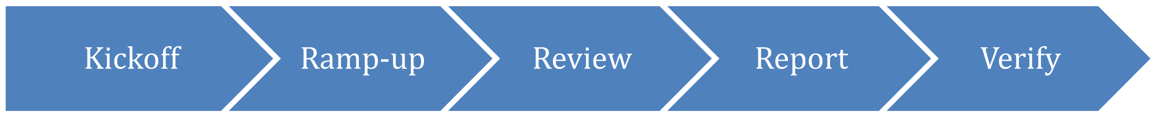

Figure 2: Methodology Flow

Kickoff

The project is kicked all of the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project as a whole. We may conclude that the overall risk is low, but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These are a solid baseline for severity determination.

The Classification of identified problems and vulnerabilities

There are four severity levels of an identified security vulnerability.

Critical – vulnerability that will lead to loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - A vulnerability that can lead to loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that can not be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Tx signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - a vulnerability that hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-revied crypto functions
- Program crashes leaves core dumps or write sensitive data to log files

Low - Problems that have a security impact but does not directly impact the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Scott Carlson	Head of Blockchain Center of Excellence	Scottj.carlson@kudelskisecurity.com