NFTSwitch – Fee Split Contract Audit Report

Prepared for NFT-Switch, 28 October 2022

Table of Contents

Table of Contents **2**

Introduction **3**

- **Scope** **3**
- **Methodologies** **4**
- **Code Criteria and Test Coverage** **4**

Vulnerabilities Summary **5**

Detailed Vulnerabilities **6**

- 1 - Any address may pass flush argument 6
- 2 - Instantiation does not prevent duplicate allocation holdings 7
- 3 - Contract should use two-step ownership transfer 8
- 4 - Revise unreachable condition 9
- 5 - A large number of Allocation Holdings may cause an out of gas error 10
- 6 - Remove unused commented code-blocks 11
- 7 - Remove no action condition and return error 12

Document control **13**

Appendices **14**

Introduction

SCV was engaged by NFTSwitch to assist in identifying security threats and vulnerabilities that have the potential to affect their security posture. Additionally, SCV will assist the team in understanding the risks and identifying potential mitigations.

Scope

SCV performed the security assessment on the following codebase:

- https://github.com/PFC-Validator/PFC-fee-split
- Code Freeze: *a7a8b41aaa3ba455136749dc048ea76119e71629*

During the audit, the following commit hash `b7a06b0ebae84f569e395c629ca1ff051b467a6b` was committed to the codebase and included in scope.

Remediations were applied up to the following committed hash/tag:

- Hash: *617b8e249eab2c7cda5ac91b660c093c2f120f3f*
- Release Tag: 0.2.1

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to NFTSwitch. Testing includes, but is not limited to, the following:

- Understanding the application and its code base purpose;
- Deploying SCV in-house tooling to automate dependency analysis and static code review;
- Analyse each line of the code base and inspect application security perimeter;
- Review underlying infrastructure technologies and supply chain security posture;

Code Criteria and Test Coverage

SCV used a scale from 0 to 10 that represents how *SUFFICIENT* or *NOT SUFFICIENT* each code criteria was during the assessment:

Criteria	Status	Notes
Provided Documentation	NOT SUFFICIENT 🔴	The codebase did not come with documentation.
Code Coverage Test	SUFFICIENT 🟢	N/A
Code Readability	SUFFICIENT 🟢	The codebase had good readability and utilised many Rust and CosmWasm best practices.
Code Complexity	SUFFICIENT 🟢	N/A

Vulnerabilities Summary

#	Summary Title	Risk Impact	Status
1	Any address may pass flush argument	**Medium**	**Resolved**
2	Instantiation does not prevent duplicate allocation holdings	**Low**	**Resolved**
3	Contract should use two-step ownership transfer	**Low**	**Resolved**
4	Revise unreachable condition	**Low**	**Resolved**
5	A large number of Allocation Holdings may cause an out of gas error	**Low**	**Acknowledged**
6	Remove unused commented code-blocks	**Informational**	**Resolved**
7	Remove no action condition and return error	**Informational**	**Resolved**

Detailed Vulnerabilities

1 – Any address may pass flush argument

Risk Impact: Medium - **Status**: Resolved

Description

The *execute_deposit* function in *PFC-fee-split/contracts/pfc-fee-splitter/src/handler/exec.rs:28* allows any caller to perform a deposit and to pass the *flush* flag. This means that any caller can perform a flush operation and push the allocated funds to each *allocation_holding*. This mechanism is problematic because it undermines the *allocation_holding.send_after* validation, which first determines whether a set threshold is met before distributing allocated funds.

Recommendations

We recommend either restricting the addresses that may perform the flush operation to a governance controlled whitelist.

2 – Instantiation does not prevent duplicate allocation holdings

Risk Impact: Low - **Status**: Resolved

Description

The `instantiate` function in `PFC-fee-split/contracts/pfc-fee-splitter/src/contract.rs:37` does not properly deduplicate the initial allocations in `msg.allocation`. The function does not enforce that each allocation name is unique, and in `PFC-fee-split/contracts/pfc-fee-splitter/src/contract.rs:78` the function does not prevent an existing allocation from being overwritten in `ALLOCATION_HOLDINGS`.

We classify this as a low risk impact due to the fact that it can only occur during the instantiation phase.

Recommendations

We recommend enforcing that each `allocation_holding` should have a unique name and if a duplicate name is encountered, then the instantiation should return an error.

3 – Contract should use two–step ownership transfer

Risk Impact: Low - **Status**: Resolved

Description

While the contract currently makes great use of the CW Admin Controller we recommend adding one additional step to ensure that admin transfers are executed securely and safely. The current ownership transfer is executed in one step, which imposes a risk that if the value of `gov_contract` in `PFC-fee-split/contracts/pfc-fee-splitter/src/handler/exec.rs:217` is incorrect, then the admin privileges of the contract are effectively transferred and lost. A two-step ownership transfer is best practice because it requires that the new admin accept ownership before the transfer and config changes actually occur.

Recommendations

We recommend implementing a two-step ownership transfer where the current admin proposes a new admin address, and then that new admin address must call the contract to accept ownership within a finite time frame.

4 – Revise unreachable condition

Risk Impact: Low - **Status**: Resolved

Description

The *`determine_allocation`* function in *`PFC-fee-split/contracts/pfc-fee-splitter/src/handler/exec.rs:257`* contains a condition that is not reachable. It is not possible that *`portion`* is zero and also greater than *`1/10_000`*. In this function the else condition will always be executed.

Recommendations

We recommend revising the condition on *`PFC-fee-split/contracts/pfc-fee-splitter/src/handler/exec.rs:257`* and also clearly documenting its purpose within a code comment.

5 – A large number of Allocation Holdings may cause an out of gas error

Risk Impact: Low - **Status**: Acknowledged

Description

Both the `execute_reconcile` and `do_deposit` deposit perform unbounded iteration on `ALLOCATION_HOLDINGS`. If the map grows too large it may cause out-of-gas errors. Within each allocation holding iteration, there is nested iteration that can quickly cause out of gas errors.

For example, within `do_deposit`, `funds_in` and `allocation_holding.balance` are also unbounded vectors that are iterated over within each allocation holding iteration. So this nested iteration can quickly become highly gas consuming.

We classify this finding as Low risk impact because the contract admin is able to remove Allocation Holdings and only the admin may add Allocation Holdings.

Recommendations

We recommend ensuring that governance is aware that `ALLOCATION_HOLDINGS` has size limitations.

6 – Remove unused commented code–blocks

Risk Impact: Informational - **Status**: Resolved

Description

The codebase contains multiple commented debug statements and unused code-blocks. It is best practice to remove all unused code comments before the codebase is used in production.

The following code comments should be removed:

- *PFC-fee-split/contracts/pfc-fee-splitter/src/contract.rs:18,28, and 134-143*
- *PFC-fee-split/contracts/pfc-fee-splitter/src/handler/exec.rs:34, 242-243 and 250-256, 269*
- *PFC-fee-split/contracts/pfc-fee-splitter/Cargo.toml:27,36,47-48 and 59-62*

Recommendations

We recommend removing the code comments mentioned above.

7 – Remove no action condition and return error

Risk Impact: Informational - **Status**: Resolved

Description

The *execute_deposit* function in *PFC-fee-split/contracts/pfc-fee-splitter/src/handler/exec.rs:19* simply returns attributes if specified message conditions are met. It is best practice to only include state modifying transactions in the execute entry points.

Recommendations

We recommend returning an error if the conditions mentioned above are met, or clearly defining a use-case for this condition in the code comments.

Document control

Version	Date	Approved by	Changes
0.1	25/10/2022	Vinicius Marino	Document Pre-Release
1.0	27/10/2022	Vinicius Marino	Revisions and Final Release

Appendices

A. Appendix – Risk assessment methodology

A qualitative risk assessment is performed on each vulnerability to determine the impact and likelihood of each.

Risk rate will be calculated on a scale. As per criteria Likelihood vs Impact table below:

	Rare	Unlikely	Possible	Likely
Critical	Medium	Severe	Critical	Critical
Severe	Low	Medium	Severe	Severe
Moderate	Low	Medium	Medium	Severe
Low	Low	Low	Low	Medium
Informational	Informational	Informational	Informational	Informational

LIKELIHOOD

- Likely: likely a security incident will occur;
- Possible: It is possible a security incident can occur;
- Unlikely: Low probability a security incident will occur;
- Rare: In rare situations, a security incident can occur;

IMPACT

- Critical: May cause a significant and critical impact;
- Severe: May cause a severe impact;
- Moderate: May cause a moderated impact;
- Low: May cause low or none impact;
- Informational: May cause very low impact or none.

B. Appendix – Report Disclaimer

The content of this audit report is provided “As is”, without representations and warranties of any kind.

The author and their employer disclaim any liability for damage arising out of, or in connection with, this audit report.