NFTswitch - NFTSwitch Contracts - Audit Report

Prepared for NFT-Switch, 11 October 2022

Table of Contents

Introduction

SCV was engaged by NFT-Switch 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/tvl83/nftswitch-dev
- Code Freeze: *7f6d937e45caa969165114ebf6213f17cad53e43*

Remediations were applied into several commits up to the following codebase and hash commit:

- https://github.com/tvl83/nftswitch-dev/tree/audit-fixes
- Code Freeze *2e789a97207158fa37f6245021e9b560584748ed*

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 NFT-Switch. 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(6-10)** or **NOT SUFFICIENT(0-5)** each code criteria was during the assessment:

Criteria	Status	Scale Range	Notes
Provided Documentation	**Sufficient**	5-6	N/A
Code Coverage Test	**Sufficient**	7-8	N/A
Code Readability	**Sufficient**	6-8	N/A
Code Complexity	**Sufficient**	6-8	N/A

Vulnerabilities Summary

	Title and Summary	Risk	Status
1	Anyone can cancel trade on behalf of other users	High	Remediated
2	Fees are calculated based on global configuration instead of stored value	High	Remediated
3	Incorrect comparison when removing stale trades allow anyone to remove trades	High	Remediated
4	Trade expiration is not considered and validated	High	Partially Remediated
5	GetFeesForTrade query returns incorrect information	Medium	Remediated
6	Incorrect buyer fee instantiation value	Medium	Remediated
7	Emergency break does not restrict removing stale trades	Low	Remediated
8	Lack of seller fee validation might cause trades to be unable to execute	Low	Remediated
9	Consider only removing stale trades based on trade and approval expiration	Informational	Acknowledged
10	Expired trades can be confirmed by the operator	Informational	Acknowledged
11	General inefficiencies in the codebase	Informational	Partially Remediated
12	Redundant statements in the codebase	Informational	Remediated

Detailed Vulnerabilities

1. Anyone can cancel trade on behalf of other users

Likelihood	Impact	Risk
Likely	Severe	**High**

Note

The team resolved this issue by only allowing the buyer or seller to cancel trades.

Description

In `src/execute.rs:165`, the `try_cancel_trade` functionality allows anyone to cancel a trade for other users. A user can simply provide the trade key combination, and the associating trade key will be removed from the storage. As a result, an attacker can perform a denial of service attack on the contract by removing all valid trades.

Recommendations

Consider only allowing authorized users to cancel trades in `try_cancel_trade`, such as the admin or the operator.

2. Fees are calculated based on global configuration instead of stored value

Likelihood	Impact	Risk
Likely	Moderate	**High**

Description

In `src/execute.rs:262,263`, the buyer and seller fees are calculated based on the config value instead of the stored value. As the buyer and seller fees are stored during the trade creation process in lines 116 and 117, the final trade should calculate the fees based on the stored value. Moreover, the `GetFeesForTrade` query functionality returns the stored fee value instead of the config fee value, which will surprise the buyer and seller when the executed amount and queried amount aren't equal.

Besides that, when the admin or operator creates a trade with 0 buyer and seller fees, as seen in `src/execute.rs:124-125`, the fees should be 0 when finalizing the trade. However, this will not be the case when executing the trade as the fees are fetched from configuration and not the trade struct itself.

Recommendations

Consider using the stored fee value to calculate the fees instead of the config value to calculate the fees. For instance, `src/execute.rs:261` should be modified into `let buyer_fee = trade.buyer_fee.clone()* trade.price.amount.clone();` while line 262 should be modified into `let seller_fee = trade.seller_fee.clone()* trade.price.amount.clone();`.

3. Incorrect comparison when removing stale trades allow anyone to remove trades

Likelihood	Impact	Risk
Likely	Severe	**High**

Description

In `src/execute.rs:359`, the `is_operator_or_admin` functionality is used to determine whether the caller is an authorized admin or operator. However, the function uses an incorrect if and equal comparison when checking the caller, as seen in `src/helpers.rs:107`. As a result, anyone other than the admin or operator is allowed to remove stale trades.

Recommendations

Consider changing `src/helpers.rs:107` into `if cfg.operator != info.sender.as_ref() || cfg.admin != info.sender.as_ref()` to correctly prevent unauthorized users to remove stale trades.

4. Trade expiration is not considered and validated

Likelihood	Impact	Risk
Likely	Moderate	High

Notes

Team implemented suggestion 1 & 3. The approval expiration returned by the contract still pending.

Description

In `src/execute.rs:79`, the seller can specify an expiration timestamp of the trade using the `expires_at` parameter. Ideally, the seller would expect the trade to expire once the expiration timestamp passes. However, this is not the case, as the contract does not perform trade expiration validation anywhere.

As a result, sellers cannot limit their trades within an expiration timestamp.

Recommendations

Consider applying the following recommendations:

1. Revert an error if the trade is expired during `try_execute_trade`.
2. Validate that the approval expiration returned in `src/execute.rs:136` is equal to or greater than the `expires_at` value to ensure the approval lasts until the end of the trade.
3. Verify the `expires_at` value supplied by the seller is in the future (ie. greater than `env.block.time()`) during `try_create_trade`.

5. GetFeesForTrade query returns incorrect information

Likelihood	Impact	Risk
Likely	Low	**Medium**

Description

In `src/query.rs:101`, the `query_get_fees_for_trade` functionality attempts to query the trade key from storage and returns the results. However, since the values returned do not correspond to the variable, it will give a misleading result to the user.

- *seller_fee* (*src/query.rs:114*)
  - *Suggestion*: `trade.as_ref().unwrap().seller_fee,`
- *seller_cost* (*src/query.rs:115*)
  - Suggestion: `Coin::new(u128::from(trade.as_ref().unwrap().price.amount * trade.as_ref().unwrap().seller_fee), NATIVE_DENOM),`
- *buyer_cost* (*src/query.rs:117*)
  - Suggestion: `Coin::new(u128::from(trade.as_ref().unwrap().price.amount * trade.as_ref().unwrap().buyer_fee), NATIVE_DENOM),`

Recommendations

Consider modifying the value according to the suggestions above.

6. Incorrect buyer fee instantiation value

Likelihood	Impact	Risk
Likely	Low	**Medium**

Description

In `src/contract.rs:24`, the buyer fee value is set to `msg.seller_fee` value. Since the seller fee value is intended only for sellers, it might cause an extra/lower fee charged to the buyer.

Recommendations

Consider changing `src/contract.rs:24` into `buyer_fee: msg.buyer_fee,`.

7. Emergency break does not restrict removing stale trades

Likelihood	Impact	Risk
Possible	Low	**Low**

Description

When removing stale trades in `src/execute.rs:353`, no validation checks the emergency brake is activated. As a result, the `try_remove_stale_trade` functionality will still work even though the admin started the emergency break.

Recommendations

Consider checking whether the emergency brake is activated in `try_remove_stale_trade` before allowing the execution to continue, such as `src/execute.rs:318-210`.

8. Lack of seller fee validation might cause trades to be unable to execute

Likelihood	Impact	Risk
Unlikely	Low	**Low**

Description

In `src/execute.rs:262,281`, a portion of the trade price is calculated and taken as seller fees. There are no validations when setting the decimal value. If a misconfiguration occurs and the seller fee gets charged over 100% (i.e., `Decimal::one()`), it would cause the execution to fail in line 281 due to an underflow error.

Affected code lines and variables:

- `src/contract.rs:24` (`msg.seller_fee`)
- `src/execute.rs:57` (`seller_fee`)
- `src/execute.rs:345` (`seller_fee_pct`)

Recommendations

Consider verifying the decimal values are below `Decimal::one()` for the code lines mentioned above.

9. Consider only removing stale trades based on trade and approval expiration

Likelihood	Impact	Risk
Unlikely	Informational	**Informational**

Note

The team mentioned that validating trades is done entirely off chain from an authorized account only. The process will run as a cron job on a regular schedule.

Description

In `src/execute.rs:353`, the `try_remove_stale_trade` functionality does not check any condition before removing a trade. This allows any trades to be removed from the storage, including valid ones.

Instead, we recommend adding checks to ensure the trade is stale before removal. For example, we recommend only removing expired trades or the approval is revoked/expired.

Recommendations

Consider verifying the trade and approval is not expired before removing stale trades in `try_remove_stale_trade`.

10. Expired trades can be confirmed by the operator

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Note

The team believes this is not an issue because an API call will verify and confirm the trade after creation. As a result, it is unlikely that an expired trade will ever be unconfirmed.

Description

In `src/execute.rs:303`, the `try_confirm_trade` functionality does not ensure the trade is not expired before confirming them. Since expired trades are stale and should be removed, we recommend adding validation to ensure only fresh trades can be confirmed by the operator.

Recommendations

Consider checking whether the trade is expired before confirming the trade in the `try_confirm_trade` functionality.

11. General inefficiencies in the codebase

Likelihood	Impact	Risk
Likely	Informational	**Informational**

Notes

Not all suggestions were implemented.

Description

In several instances in the codebase, some inefficiencies can be modified to reduce the code complexity and overall gas consumption. These do not introduce any security risk but can be refactored to increase the readability and maintainability of the codebase as per description below:

- listing_fee(), price() (*src/execute.rs:98, src/contract.rs:25, src/execute.rs:61)*
  - Issue: The contract assumes all payments are transacted in `uluna`, hence there's no need to provide the denom value.
  - Suggestion: Instead of providing the whole `Coin` struct, consider using `amount` directly
- try_remove_stale_trade() (*src/execute.rs:359-363*)
  - Issue: As `is_operator_or_admin` should error for an invalid caller, there is no need to return a boolean value.
  - Suggestion: Modify `is_operator_or_admin` to return StdResult and remove the boolean check.
- try_confirm_trade() (*src/execute.rs:310,334-336*)
  - Issue: An operator can remove trades using `CancelTrade`, the remove functionality can be removed from `try_confirm_trade`.
  - Suggestion: Remove `is_confirmed_by_operator` argument, assume operator confirms a trade when they call `try_confirm_trade`. The operator can call CancelTrade if a trade should be removed.

Recommendations

Consider following the suggestions mentioned above.

12. Redundant statements in the codebase

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

In several instances of the codebase, the conditional statements can be modified to improve the overall performance.

Firstly, the if statement in `src/execute.rs:93` verifies the amount sent by the user equals the listing fee amount. If the listing fee configured by the admin is 0, it will break this functionality as Cosmos SDK prevents 0 amount from being sent.

Secondly, the else if statement in `src/execute.rs:203` can be removed because the contract only adds the buyer into the `TradeKey` struct instead of the seller. Even if a seller will be supported in the future, it is possible to cancel the trades by replacing the buyer's address with the seller's.

Thirdly, the if statement in `src/execute.rs:190-192` can be removed directly because `trade.buyer` is retrieved from the `buyer_addr` argument. Hence, it is already validated when saving and loading from the trade storage state.

Fourthly, the code complexity in `try_execute_trade` can be reduced by removing the `buyer` string argument. Since the buyer is only expected to call `the function, the checks in` src/execute.rs:244, 255-258‘ can be removed.

Lastly, the if condition in `src/execute.rs:280-282` is redundant because the statement only checks if the `amount_send` exceeds 0 while the amount used is `amount_send.sub(commission)`. The execution will fail if the final amount is 0, as Cosmos SDK doesn't allow 0 number of funds to be transferred.

Recommendations

1. Check whether the user sent funds if the listing fee amount exceeds 0 (i.e. wrap `src/execute.rs:91-95` into `if !cfg.listing_fee.amount.is_zero(){}`)
2. Remove the else if statement in `src/execute.rs:203-225`, remove the `seller` argument, and modify the `buyer` variable type into `String`.
3. Remove `src/execute.rs:190-192`.
4. Remove `buyer` argument from `try_execute_trade`, and load the trade key in `src/execute.rs:248` using `trade_key(&info.sender, &nft_collection, nft_id.clone())`.
5. Modify `src/execute.rs:280` into `if !amount_send.sub(commission).is_zero()`.

Document control

Document changes

Version	Date	Name	Changes
0.1	2022-09-29	Vinicius Marino	Initial report
0.2	2022-09-30	Vinicius Marino	Team communication and Pre-Release
1.0	2022-10-11	Vinicius Marino	Revisions and Document Release

Document contributors

Name	Role	Email address
Vinicius Marino	Security Specialist	vini@scv.services

Appendices

Appendix A: 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.

Copyright of this report remains with the author.

Appendix B: 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:

Likelihood / Impact	Rare	Unlikely	Possible	Likely
Critical	**Medium**	**High**	**Critical**	**Critical**
Severe	**Low**	**Medium**	**High**	**High**
Moderate	**Low**	**Medium**	**Medium**	**High**
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.

SCV