Wicca Money - Wicca Launchpad Contracts - Audit Report

Prepared for Wicca Money, 25 July 2022

Table of Contents

Introduction

SCV was engaged by Wicca Money 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/wicca-money/wicca-contract-audit

The following contracts were in-scope with the following code hash *287652007f73a39d0670efcb8b3465a598d95eaa*:

- token minter
- token vesting (locker)
- launchpad

Remediations were applied into several commits PRs which were verified by SCV.

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 Wicca Money. 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**	7-8	N/A
Code Coverage Test	**Sufficient**	7-8	N/A
Code Readability	**Sufficient**	6-8	N/A
Code Complexity	**Sufficient**	6-7	N/A

Vulnerabilities Summary

	Title and Summary	Risk	Status
1	Consider enforcing validations for dex factory address	**Medium**	**Remediated**
2	Recharge fee might be inconsistent if the admin updates the fixed asset	**Medium**	**Remediated**
3	Consider verifying the claim percentage value in the locker contract	**Low**	**Remediated**
4	Extra funds sent when buying launchpad tokens are lost	**Low**	**Remediated**
5	Lack of validation in token factory contract during contract instantiation and update config	**Low**	**Remediated**
6	Loose validations in launchpad contract during contract instantiation and update config	**Low**	**Remediated**
7	Consider allowing project owners to have the option of minting tokens	**Informational**	**Acknowledged**
8	Consider favoring default blacklist approach for sale status	**Informational**	**Remediated**
9	Excessive whitelisted currencies would cause currency whitelist query message to fail	**Informational**	**Acknowledged**
10	Fee asset can be configured as CW20 token that ignores fee requirements from users	**Informational**	**Remediated**
11	Listing discount validation can be refactored	**Informational**	**Remediated**
12	Possible rounding issue leads users to buy zero tokens	**Informational**	**Remediated**
13	Token sale whitelists can still be updated for canceled launchpad	**Informational**	**Remediated**
14	Typographic error in function and variable name	**Informational**	**Remediated**

Detailed Vulnerabilities

1. Consider enforcing validations for dex factory address

Likelihood	Impact	Risk
Unlikely	Moderate	**Medium**

Description

In *contracts/launchpad/src/dex/mod.rs:188* and *245*, the contract creates the asset pair if the factory query message fails. As there are no checks in place to verify that the error received is actually *"pool pair not found"*, an incorrect contract address being set as dex factory address will also enter this error field. Since the sub-message configured will only reply on success, the users or admin will only notice that the pool creation failed after noticing there are no vaults created in the locker contract.

Even worse, the admin wouldn't be able to cancel the sale forcefully since the sale owner or user already redeemed their tokens or funds. As a result, the funds that should be used to create and add liquidity to the pool would be stuck in the contract.

Recommendations

Consider verifying the dex factory address to correct by querying the factory's configurations, this reduces the likelihood of an incorrect address being set.

2. Recharge fee might be inconsistent if the admin updates the fixed asset

Likelihood	Impact	Risk
Possible	Moderate	**Medium**

Description

In *contracts/launchpad/src/handler.rs:124-127*, users can recharge their purse balance in the `FEE_PURSE` state by sending additional funds when calling the `RechargeFee` message.

We have discovered two issues in this function. Firstly, although any denoms sent by the user are recorded by `FEE_PURSE`, the funds cannot be withdrawn. `FEE_PURSE` is used to charge the user a fixed asset fee as defined by the contract as seen in *contracts/launchpad/src/handler.rs:154-162*. If the user sent any other funds than the configured fixed fee asset, the funds are stuck in the contract.

Secondly, if the users pre-recharge their purses in the contract, the admin can simply update the fixed fee asset to another denom, which renders the user's recharged fee useless. This is because the user needs to recharge their fees with the updated fixed asset denom, and their old funds will be stuck in the contract until the admin decides to switch the funds back.

Recommendations

Consider allowing the users to withdraw deposited funds inside `FEE_PURSE` and additionally limiting their deposits to match the fixed fee asset configured by the admin.

3. Consider verifying the claim percentage value in the locker contract

Likelihood	Impact	Risk
Unlikely	Low	**Low**

Description

When instantiating or updating configuration in the locker contract, the decimal value of `msg.claim_percentage_fee` is not validated to be lower or equal to *1.0*. A misconfiguration of this value would cause an overflow error in *contracts/locker/src/handler.rs:372*, causing users unable to claim their tokens.

Recommendations

Consider verifying the decimal value is configured equal to or below 1.0 in *contracts/locker/src/contract.rs:47* and *contracts/locker/src/handler.rs:30*.

4. Extra funds sent when buying launchpad tokens are lost

Likelihood	Impact	Risk
Rare	Low	**Low**

Description

When a user tries to buy launchpad tokens in *contracts/launchpad/src/contract.rs:154*, only the first index of the funds sent by the user is recorded. If the user sent additional funds, the funds would be stuck in the contract and unable to be taken.

This issue is also present in *contracts/token-factory/src/contract.rs:127* when creating tokens.

Recommendations

Consider validating funds to match with expected funds to be received.

5. Lack of validation in token factory contract during contract instantiation and update config

Likelihood	Impact	Risk
Unlikely	Low	**Low**

Description

When instantiating the token factory contract, several configuration parameters are not validated. For example, the `msg.percentage_fee` parameter should be validated below 1.0 decimal value, and the `msg.code_id` value should be validated higher than 0.

Affected code lines: - contracts/token-factory/src/contract.rs:44 - contracts/token-factory/src/contract.rs:52 - contracts/token-factory/src/contract.rs:89 - contracts/token-factory/src/contract.rs:93

Recommendations

Consider implementing the validations as mentioned.

6. Loose validations in launchpad contract during contract instantiation and update config

Likelihood	Impact	Risk
Possible	Low	**Low**

Description

When instantiating the launchpad contract, several configuration parameters are loosely validated. For example, the `msg.min_cap` parameter in *contracts/launchpad/src/contract.rs:51* is not validated that the second index is larger to be the first index and both values are larger than 0. If a misconfiguration happens, users will be unable to sell their tokens due to assertion errors in *contracts/launchpad/src/sale.rs:170* and *174*.

Note that some of the issues also applied when updating the configuration via the *UpdateConfig* message.

Recommendations

Consider applying the following validations in *contracts/launchpad/src/contract.rs*:

- `msg.min_cap[1] >= msg.min_cap[0] && msg.min_cap[0] > Uint128::zero()`
- `msg.min_token_sale_amt > Uint128::zero()`
- `msg.token_sale_pct_fee <= Decimal::one()&& msg.raised_cur_pct_fee <= Decimal::one()` *(issue also present in line 109 and 113)*
- Verify no duplicate keys in `msg.dex_addr` parameter as it would cause the `DEX_FACTORY` state to be overwritten (issue also present in `execute_update_dex_factory`)

7. Consider allowing project owners to have the option of minting tokens

Likelihood	Impact	Risk
Unlikely	Informational	**Informational**

Description

In *contracts/token-factory/src/contract.rs:173*, the minter of the tokens is set to None. This means that the token owners will have no option to mint additional tokens if necessary.

Recommendations

Consider allowing the users to have the option of setting the token minter as themselves.

8. Consider favoring default blacklist approach for sale status

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

In *contracts/launchpad/src/sale.rs:284*, the sale status by default enters an ended state instead of a failed state. This approach of default whitelist strategy is cumbersome to maintain as the if statement must make sure to catch all comparisons that would render the sale status as failed. If there's a comparison that fails to set the status as failed, it would cause the sale status to incorrectly returned as ended, which is incorrect.

Recommendations

Consider using a "default blacklist" approach which returns the sale status to failed by default. Instead of finding all possible "failed" approaches, the contract can simply focus on finding all "successful" approaches. As there is a maximum number of "successful" approaches, it is easier to manage than finding all "failed" approaches. This can be done by modifying lines *276* to *278* to return the sale status as ended with correct comparisons and returning the default status as failed.

```
if now >= self.end && progress.cur_raised >= self.soft_cap {
            return SaleStatus::Ended;
        }

(...)

SaleStatus::Failed
```

9. Excessive whitelisted currencies would cause currency whitelist query message to fail

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

The `query_currency_whitelist` query message in *contracts/launchpad/src/querier.rs:29-36* attempts to loop over all whitelisted denom and token currencies without pagination. If there are too many denoms or token currencies whitelisted by the admin, the query message would fail due to an out-of-gas error.

With that said, the admin can prevent this issue from happening by removing some of the currencies via `UpdateCurrencyWhitelist` execute message.

Recommendations

Consider paying attention to the number of currencies to be whitelisted to prevent unexpected query message failure.

10. Fee asset can be configured as CW20 token that ignores fee requirements from users

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

When instantiating fixed fee assets, the `AssetInfoUnchecked` supplied can be either *CW20* token address or native token assets. A misconfiguration of fixed fee assets as *CW20* token address would cause the protocol not taking any fees from the user. As seen in *contracts/launchpad/src/handler.rs:148-166*, the fee is only charged from the user if the fixed fee asset is configured as native token assets.

This issue is also present in the following code lines:

- *contracts/launchpad/src/contract.rs:44*
- *contracts/launchpad/src/handler.rs:104*
- *contracts/token-factory/src/contract.rs:34*
- *contracts/token-factory/src/contract.rs:83*

Recommendations

Consider validating that the fixed fee asset supplied is a native token asset. If there's a situation where the fee is not needed to be charged by the user, the amount of the fee can be updated to 0.

11. Listing discount validation can be refactored

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

When verifying the liquidity discount decimal value in *contracts/launchpad/src/sale.rs:112*, the comparison attempts to verify the listing discount ranges from 0-50%. Since the Decimal is used in `self.listing_discount` uses an unsigned integer value for all implementations, there is no need to check the value needs to be positive. This reduces computation power and improves the efficiency of the contract.

Recommendations

Consider removing the first comparison check `(self.listing_discount >= Decimal::zero())`.

12. Possible rounding issue leads users to buy zero tokens

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

In *contracts/launchpad/src/handler.rs:281-283*, the buy function attempts to calculate the number of tokens the user bought based on the tokens they sent. Floating numbers are automatically rounded off, hence it is possible that the user receives zero tokens.

For example, imagine a project decides to raise 1000 payment tokens (eg. USDC) while having 500 project tokens for sale. With the payment tokens as 6 decimals and project tokens as 0 decimals in mind, a user that sends 1 payment token would receive 0 project tokens *(1_000_000* 500 / 1_000_000_000 = 0.5 rounded to 0)*. However, this scenario requires the sales owner to have a specific configuration for their sale requirements, which is unlikely.

Recommendations

Consider verifying the token bought amount is more than 0 (`token_bought_amt > Uint128::zero()`). If not, revert an error to the user.

13. Token sale whitelists can still be updated for canceled launchpad

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

The `execute_update_whitelist` function in *contracts/launchpad/src/handler.rs:205* allows the sale owner to update participant whitelists even if the sale is forced canceled by the admin. This might give a false intention to people that the sale is still live.

Recommendations

Consider preventing the sale owner to update whitelists if their sale is canceled.

14. Typographic error in function and variable name

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

The `execute_force_cancle_liq` function in *contracts/launchpad/src/handler.rs:652* contains a typographic error in their function name which should be execute_force_cancel_liq. Other than that, the variable `force_cancle_liq` in *contracts/launchpad/src/sale.rs:226* should be `force_cancel_liq`
.

This impacts the readability of the codebase.

Recommendations

Consider correcting the typographic errors above.

Document control

Document changes

Version	Date	Name	Changes
0.1	2022-07-19	Vinicius Marino	Initial report
0.2	2022-07-20	Vinicius Marino	Team communication and Pre-Release
1.0	2022-06-25	Vinicius Marino	Document Release

Document contributors

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

Appendices

Appendix A: Report Disclaimer

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:

Impact \ Likelihood	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.