Phoenix Finance - Phoenix Core Contracts - Audit Report

Prepared for Phoenix Protocol, 17 June 2022

Table of Contents

Introduction **3**

Scope . . . 3

Methodologies . . . 4

Code Criteria and Test Coverage . . . 4

Vulnerabilities Summary **5**

Detailed Vulnerabilities **6**

1. Single admin can overwrite ADMIN_LIST . . . 6
2. Pair contract admins will be inconsistent during factory contract updates . . . 7
3. Extra funds sent are lost . . . 8
4. Queries may fail if too many pair configs are initialized . . . 9
5. Reverse simulation returns a lower offer amount than expected . . . 10
6. Consider reverting an error when the returned amount is zero . . . 13
7. Consider validating unique addresses and empty arrays . . . 14
8. Duplicate Function call . . . 15
9. Overflow checks are not enabled . . . 16
10. Remove incorrect branding . . . 17
11. Remove unnecessary address validation in the reply handler . . . 18
12. Remove unnecessary lowercase address conversions . . . 19
13. Submessages are all accepted without filtering the expected identifier value . . . 20
14. Swap function performs extra queries . . . 21
15. Typographic error found in the codebase . . . 22
16. Unnecessary pair information can be removed . . . 23
17. Unused functionality comments are not removed . . . 24

Document control **25**

Appendices **26**

Appendix A: Report Disclaimer . . . 26

Appendix B: Risk assessment methodology . . . 27

Introduction

SCV was engaged by Phoenix Protocol 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/miranafi/phoenix-core

Code freeze hash: *d1b4832db9292496f7a3909f0d226a91a203005c*

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 Phoenix Protocol. 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	Single admin can overwrite ADMIN_LIST	**High**	**Pending**
2	Pair contract admins will be inconsistent during factory contract updates	**Medium**	**Pending**
3	Extra funds sent are lost	**Low**	**Pending**
4	Queries may fail if too many pair configs are initialized	**Low**	**Pending**
5	Reverse simulation returns a lower offer amount than expected	**Low**	**Pending**
6	Consider reverting an error when the returned amount is zero	**Informational**	**Pending**
7	Consider validating unique addresses and empty arrays	**Informational**	**Pending**
8	Duplicate Function call	**Informational**	**Pending**
9	Overflow checks are not enabled	**Informational**	**Pending**
10	Remove incorrect branding	**Informational**	**Pending**
11	Remove unnecessary address validation in the reply handler	**Informational**	**Pending**
12	Remove unnecessary lowercase address conversions	**Informational**	**Pending**
13	Submessages are all accepted without filtering the expected identifier value	**Informational**	**Pending**
14	Swap function performs extra queries	**Informational**	**Pending**
15	Typographic error found in the codebase	**Informational**	**Pending**
16	Unnecessary pair information can be removed	**Informational**	**Pending**
17	Unused functionality comments are not removed	**Informational**	**Pending**

Detailed Vulnerabilities

1. Single admin can overwrite ADMIN_LIST

Likelihood	Impact	Risk
Possible	Severe	**High**

Description

The *execute_update_admins* function in *phoenix-core/contracts/whitelist/src/contract.rs:90* allows any admin to overwrite the entire `ADMIN_LIST`. This is problematic if any of the admins were to get compromised or introduce a misconfiguration accidentally.

Recommendations

It is best practice to handle the admin update in a more granular fashion, for example, adding an `add_admin`/`remove_admin` function to handle each request rather than overwriting the existing list individually.

2. Pair contract admins will be inconsistent during factory contract updates

Likelihood	Impact	Risk
Possible	Moderate	**Medium**

Description

The `execute_create_pair` function in *contracts/factory/src/contract.rs:298* instantiates a new pair contract with the admin set to `config.owner`. This may be problematic as `config.owner` is an updatable value which may lead to inconsistencies within the pair contract's admins. This will create a situation where the various pair contracts will have different admins from each other and the factory contract owner.

The main impact of this issue is that the pair contracts will become more difficult or impossible to maintain as they will have different admin. This also presents a security concern if one or more of the admins were to get lost or compromised; it creates a larger attack surface for the various pair contracts.

Recommendations

We recommend introducing validation logic where the pair contracts check that the pair owner is equal to the factory owner.

3. Extra funds sent are lost

Likelihood	Impact	Risk
Rare	Low	**Low**

Description

The `assert_sent_native_token_balance` function in *phoenix-core/packages/phoenix/src/asset.rs:79* does not correctly handle the scenario when receiving a deposit message with multiple funds.

`MessageInfo.funds` is a vector of Coin. The current logic does not account for a funds vector with a length greater than 1. If additional funds are sent to this function, they will be lost in the contract.

Recommendations

We recommend adding validation to ensure that the length of `info.funds` is equal to one and returning an error if this condition is false.

4. Queries may fail if too many pair configs are initialized

Likelihood	Impact	Risk
Unlikely	Low	**Low**

Description

In *phoenix-core/contracts/factory/src/contract.rs:477*, the `PAIR_CONFIGS` storage is fetched without a pagination limit. If the contract owner created many pair configurations with custom pair types, the `query_config` and `query_blacklisted_pair_types` query messages would fail due to exceeding the gas limit. Moreover, this will be irrecoverable since there's no entry point for the contract owner to remove `PAIR_CONFIGS` from the storage.

Recommendations

We recommend applying pagination to both `query_config` and `query_blacklisted_pair_types` query messages.

5. Reverse simulation returns a lower offer amount than expected

Likelihood	Impact	Risk
Unlikely	Low	**Low**

Description

The query_reverse_simulation query message returns a lower offer amount than expected due to a rounding issue in *phoenix-core/contracts/pair/src/contract.rs:#1062-1078*. The following test case reproduces this scenario:

```
#[test]
fn rounding_poc() {
    // test case reproduced in phoenix-core/contracts/pair/src/testing.rs

    // setup phase
    let asset_pool_amount = Uint128::new(30_000_000_000_u128);
    let collateral_pool_amount = Uint128::new(20_000_000_000_u128);
    let offer_amount = Uint128::new(1_500_000_000_u128);

    let mut deps = mock_dependencies(&[Coin {
        denom: "uusd".to_string(),
        amount: collateral_pool_amount,
    }]);

    // pre-apply user offer amount
    deps.querier.with_token_balances(&[(
        &String::from("lunax"),
        &[(
            &String::from(MOCK_CONTRACT_ADDR),
            &(asset_pool_amount + offer_amount),
        )],
    )]);

    // instantiate address
    let msg = InstantiateMsg {
        asset_infos: [
            AssetInfo::NativeToken {
                denom: "uusd".to_string(),
            },
            AssetInfo::Token {
                contract_addr: Addr::unchecked("lunax"),
            },
        ],
        token_code_id: 10u64,
        factory_addr: String::from("factory"),
        init_params: None,
```

```
    };

    let env = mock_env();
    let info = mock_info("lunax", &[]);
    let _res = instantiate(deps.as_mut(), env, info, msg).unwrap();

    // 1. we calculate how much users will receive if they offer_amount

    // Current price is 1.5, so expected return without spread is 1000
    // 952380952,3809524 = 20000000000 - (30000000000 * 20000000000) /
       (30000000000 + 1500000000)
    let expected_ret_amount = Uint128::new(952__380_952u128);

    // 47619047 = 1500000000 * (20000000000 / 30000000000) -
       952380952,3809524
    let expected_spread_amount = Uint128::new(47__619_047_u128);

    let expected_commission_amount = expected_ret_amount.multiply_ratio(3
       u128, 1000u128); // 0.3%
    let expected_return_amount = expected_ret_amount
        .checked_sub(expected_commission_amount)
        .unwrap();

    // Return asset token balance as normal
    deps.querier.with_token_balances(&[(
        &String::from("lunax"),
        &[(&String::from(MOCK_CONTRACT_ADDR), &(asset_pool_amount))],
    )]);

    // verify with swap simulation response
    let simulation_res: SimulationResponse = query_simulation(
        deps.as_ref(),
        Asset {
            amount: offer_amount,
            info: AssetInfo::Token {
                contract_addr: Addr::unchecked("lunax"),
            },
        },
    )
    .unwrap();

    // 2. we assert that swap simulation returns the correct amount
    assert_eq!(expected_return_amount, simulation_res.return_amount);
    assert_eq!(expected_commission_amount, simulation_res.
       commission_amount);
    assert_eq!(expected_spread_amount, simulation_res.spread_amount);

    // 3. we try compare the results with reverse swap simulation
    let reverse_simulation_res: ReverseSimulationResponse =
       query_reverse_simulation(
        deps.as_ref(),
        Asset {
            amount: expected_return_amount,
            info: AssetInfo::NativeToken {
```

```
                denom: "uusd".to_string(),
            },
        },
    )
    .unwrap();

    // 4. due to a rounding issue, the reverse simulation response returns
        a lower amount than the original offer amount
    assert_ne!(reverse_simulation_res.offer_amount, offer_amount);

    assert_eq!(reverse_simulation_res.spread_amount,
       expected_spread_amount);
    assert_eq!(
        reverse_simulation_res.commission_amount,
        expected_commission_amount
    );

    // 5. assume the user uses the incorrect result from reverse
       simulation response
    let simulation_res: SimulationResponse = query_simulation(
        deps.as_ref(),
        Asset {
            amount: reverse_simulation_res.offer_amount,
            info: AssetInfo::Token {
                contract_addr: Addr::unchecked("lunax"),
            },
        },
    )
    .unwrap();

    // 6. user will receive a lower return amount than expected
    assert_ne!(expected_return_amount, simulation_res.return_amount);
}
```

Recommendations

We recommend incrementing the offer amount value by one in line *1088*, so the returned offer amount is correct.

6. Consider reverting an error when the returned amount is zero

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

When executing a swap message operation in *phoenix-core/contracts/router/src/operations.rs:#65-72*, there's a chance that the amount returned is zero. This might happen when the user provided a duplicate swap operation or the user did not send any tokens when using this functionality. The current implementation will continue executing even if there's no amount being swapped, which is inefficient.

Recommendations

We recommend reverting an error if the amount returned is zero in line *#65*.

7. Consider validating unique addresses and empty arrays

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

When instantiating and updating admins in the whitelist contract, the validate_addresses function is called to verify the provided admin array contains valid addresses. However, the current implementation does not remove duplicate addresses and ensures the array is not empty. The former would cause inefficiencies in the contract, while the latter would cause the `ADMIN_LIST` storage state to be empty. As a result, the contracts will be locked forever, and no one will be able to use them, although the contract configuration is still mutable.

Recommendations

We recommend validating unique addresses and making sure the provided array is not empty.

8. Duplicate Function call

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

The `execute_create_pair` function calls the *pair_key* function twice per invocation, which will impact efficiency. This occurs in *contracts/factory/src/contract.rs:314* and *328*. This function can call pair_key once and store the value rather than calculating the *pair_key* twice.

Recommendations

We recommend moving line *contracts/factory/src/contract.rs:328* to before line *#314* and using the stored pair key rather than calling it again.

9. Overflow checks are not enabled

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

The following packages do not have overflow-checks enabled in their manifest file:

- phoenix-core/contracts/factory/Cargo.toml
- phoenix-core/contracts/pair/Cargo.toml
- phoenix-core/contracts/pair_stable/Cargo.toml
- phoenix-core/contracts/periphery/oracle/Cargo.toml
- phoenix-core/contracts/router/Cargo.toml
- phoenix-core/contracts/token/Cargo.toml
- phoenix-core/contracts/whitelist/Cargo.toml

Recommendations

Although integer overflow checks are enabled at the workspace level, we still recommend enabling them in each package to prevent issues introduced by future code refactoring. This can be done by specifying overflow-checks as true in manifest profiles.

10. Remove incorrect branding

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

The codebase contains some branding that has not been changed from the original Astroport branding, which should be changed to Phoenix. The following are instances that should be changed:

- *phoenix-core/contracts/factory/src/contract.rs:457*
- *phoenix-core/packages/phoenix/src/factory.rs:55*

Recommendations

We recommend replacing the following instances of Astroport with Phoenix.

11. Remove unnecessary address validation in the reply handler

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

The factory's reply entry point performs an unnecessary validation on *pair_contract* at *phoenix-core/contracts/factory/src/contract.rs:379*.

This validation is unnecessary because this address is being returned directly from the newly instantiated contract.

Recommendations

We recommend removing the validation being performed on *phoenix-core/contracts/factory/src/contract.rs:379* and directly storing the value returned from *res.get_contract_address().*

12. Remove unnecessary lowercase address conversions

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

There are various instances in the contracts where addresses are converted to lowercase during validation. This extra conversion is no longer necessary where `addr_validate` from *cosmwasm_std* is already in use, as the function has been updated to include an address case validation.

Performing this conversion multiple times consumes unnecessary computation and gas usage, which is inefficient. This can be improved by removing the duplicate check, specifically when invoking the `addr_validate_to_lower` function.

Recommendations

We recommend removing any duplicate address validation checks where `addr_validate` is already present.

13. Submessages are all accepted without filtering the expected identifier value

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

The following contracts do not restrict the reply identifier to be the expected `INSTANTIATE_PAIR_REPLY_ID` or `INSTANTIATE_TOKEN_REPLY_ID` value:

- phoenix-core/contracts/factory/src/contract.rs:367
- phoenix-core/contracts/pair/src/contract.rs:116
- phoenix-core/contracts/pair_stable/src/contract.rs:139

It is best practice to utilize a reply handler. The contracts currently designate reply ids for each action during the submessage but do not handle them in the reply entrypoint.

Recommendations

We recommend validating the reply identifier as the expected constant value and rejecting any unexpected reply handlers entering the sub-message entry points.

14. Swap function performs extra queries

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

The swap function in *phoenix-core/contracts/pair_stable/src/contract.rs:#717-761* calls the `query_token_precision` function twice for each asset which will result in excess `TokenInfo` queries.

These queries can be made once per asset to improve efficiency.

Recommendations

We recommend only performing the query once per asset to improve efficiency.

15. Typographic error found in the codebase

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

In *phoenix-core/contracts/pair/src/error.rs:14*, the NonSupported error message has a typographic mistake of “non”. Ideally, it should be corrected to “not”.

Recommendations

We recommend correcting the typographic error mentioned above.

16. Unnecessary pair information can be removed

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

In *phoenix-core/contracts/periphery/oracle/src/state.rs:39*, the Config struct holds an `asset_infos` array to record the assets in the pool. The array can remove this because the `PairInfo` struct in line *#41* already contains the asset information as seen in *phoenix-core/packages/phoenix/src/asset.rs:223*.

Recommendations

We recommend removing the `asset_infos` array from the Config struct and use the `PairInfo` struct's asset information instead.

17. Unused functionality comments are not removed

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Description

As a fork from the Astroport contract, some comments should be deleted along with the removed functionality. In *phoenix-core/contracts/router/src/contract.rs:80*, the `AssertMinimumReceive` execute message was removed from the fork, but the comments and the associated information in the README file are still preserved. Other than that, the comment in *phoenix-core/packages/phoenix/src/router.rs:80* is empty.

Recommendations

We recommend removing the unrelated comments and completing the missing comment mentioned above.

Document control

Document changes

Version	Date	Name	Changes
0.1	2022-06-15	Vinicius Marino	Initial report
0.2	2022-06-17	Vinicius Marino	Team communication and Pre-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.

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.


SCV