Smart contract security audit VaporFi

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During March of 2023, VaporFi team engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. VaporFi provided CTDSec with access to their code repository and whitepaper.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that’s able to withstand the network’s fast-paced and rapidly changing environment, we at CTDSec recommend that VaporFi team put in place a bug bounty program to encourage further and active analysis of the smart contract.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the VaporFi contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source code:

https://github.com/VaporFi/liquid-mining/commit/4d0334045684cc7899ed2c10a83a8c2fd1556ce0

https://github.com/VaporFi/clouds/tree/main/src

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	PASSED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	MEDIUM ISSUES
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED
16	Uninitialized storage pointers.	PASSED

17	Arithmetic accuracy.	PASSED
18	Design Logic.	PASSED
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	PASSED

3.0 Security Issues

3.1 High severity issues [0]

No high severity issues found.

3.2 Medium severity issues [1]

1. SWC-101. Integer Overflow and Underflow.

Contracts affected: RestakeFacet.sol & UnlockFacet.sol

Function: _applyPoints()

Issue: An integer overflow occurs when attempting to re-stake before the creation of a new season while calculating the value of daysUntilSeasonEnd at the end of a season.

Solution: Check s.seasons[_seasonId].endTimestamp > block.timestamp.

3.3 Low severity issues [1]

1. Wrong timestamp input not controlled

Contract: DiamondManagerFacet.sol

Function: setSeasonEndTimestamp()

Issue: An incorrect timestamp can lead to erroneous contract behavior.

Solution: Check s.seasons[_seasonId].endTimestamp > block.timestamp.

3.4 Informational issues [1]

1. Timestamp can lead to manipulation

Contract: BoostFacet.sol, DepositFacet.sol, RestakeFacet.sol y WithdrawFacet.sol

Function: claim()

Issue: The contract's behavior could be negatively impacted if the condition that verifies s.seasons[seasonId].endTimestamp >= block.timestamp & uint256 _daysUntilSeasonEnd = (s.seasons[_seasonId].endTimestamp - block.timestamp) / 1 days; is manipulated.

Solution: We recommend not using block.timestamp as a source of entropy because it can be manipulated.

4.0 Testing coverage - python

During the testing phase, custom use cases were written to cover all the logic of contracts in python language.

VaporFi tests

Boostfacet:

```python
from brownie import (
    reverts,
    BoostFacet, DepositFacet,
    DiamondManagerFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    error_hex_code
)

from scripts.deploy import (
    deploy_liquid_mining_diamond
)

def test_claim_boost(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond = 
deploy_liquid_mining_diamond(owner)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address, 
DepositFacet.abi)
    boost_facet = Contract.from_abi("BoostFacet", diamond.address, 
BoostFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet", 
diamond.address, DiamondManagerFacet.abi)
```

```
    # Asserts
    with reverts(error_hex_code('BoostFacet__UserNotParticipated()')):
        boost_facet.claimBoost(1, {"from": owner})

    # Set season
    diamond_manager.setCurrentSeasonId(1, {"from": owner})
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(5), {"from":
owner})
    # deposit
    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})

    with reverts(error_hex_code('BoostFacet__InvalidBoostLevel()')):
        boost_facet.claimBoost(1, {"from": owner})

    boost_facet.claimBoost(0, {"from": owner})
    assert diamond_manager.getUserPoints(owner, 1)[1] == 0
    with reverts(error_hex_code('BoostFacet__BoostAlreadyClaimed()')):
        boost_facet.claimBoost(1, {"from": owner})

def test_claim_boost_stratosphere(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    boost_facet = Contract.from_abi("BoostFacet", diamond.address,
BoostFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    # Set season
    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    stratosphere.setStratosphereMemberBasic(other)
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(5), {"from":
owner})
    # deposit
    amount = 100
    boost_fee = 2 * 1e6
    deposit_token.mint(other, amount)
    boost_fee_token.mint(other, boost_fee)
    deposit_token.approve(deposit_facet.address, amount, {"from": other})
    boost_fee_token.approve(boost_facet.address, boost_fee, {"from":
other})
    deposit_facet.deposit(amount, {"from": other})

    boost_facet.claimBoost(1, {"from": other})
    assert diamond_manager.getUserPoints(other, 1)[1] == 9
```

ClaimFacet:

```
from brownie import (
    reverts,
    ClaimFacet, DepositFacet,
    DiamondManagerFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    evm_increase_time,
    error_hex_code
)

from scripts.deploy import (
    deploy_liquid_mining_diamond
)

def test_claim_boost(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    claim_facet = Contract.from_abi("ClaimFacet", diamond.address,
ClaimFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    # Asserts
    # Set season
    diamond_manager.setSeasonEndTimestamp(0, get_timestamp(3), {"from":
owner}) # force season in progress
    with reverts(error_hex_code('ClaimFacet__InProgressSeason()')):
        claim_facet.claim({"from": owner})

    # incresea time 5 days
    evm_increase_time(86400 * 5)
    with reverts(error_hex_code('ClaimFacet__NotEnoughPoints()')):
        claim_facet.claim({"from": owner})

def test_claim_boost_with_deposit(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    claim_facet = Contract.from_abi("ClaimFacet", diamond.address,
ClaimFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner})
    # deposit
    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})

    # Asserts
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(3), {"from":
owner})
    # incresea time 5 days
    evm_increase_time(86400 * 5)
    # Add some mint to get reward
    reward_token.mint(claim_facet.address, 1000)
    claim_facet.claim({"from": owner})
    with reverts(error_hex_code('ClaimFacet__AlreadyClaimed()')):
        claim_facet.claim({"from": owner})
```

DepositFacet:

```
from brownie import (
    reverts,
    DepositFacet,
    DiamondManagerFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    error_hex_code
)

from scripts.deploy import (
    deploy_test_erc20,
    deploy_stratosphere_mock,
    deploy_liquid_mining_diamond
)

def test_deposit(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    # Asserts
    with reverts(error_hex_code('DepositFacet__NotEnoughTokenBalance()')):
        deposit_facet.deposit(100, {"from": owner})

    deposit_token.mint(owner, 150)
    with reverts(error_hex_code('DepositFacet__SeasonEnded()')):
        deposit_facet.deposit(100, {"from": owner})

    # Set season
    diamond_manager.setCurrentSeasonId(1, {"from": owner})
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(5), {"from":
owner})
    # deposit
    amount = 100
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})

    assert diamond_manager.getDepositAmountOfUser(owner, 1) == 95
    assert diamond_manager.getDepositPointsOfUser(owner, 1) == 95 * 5
    assert diamond_manager.getTotalDepositAmountOfSeason(1) == 95
    assert diamond_manager.getTotalPointsOfSeason(1) == 95 * (5)

    diamond_manager.setCurrentSeasonId(2, {"from": owner})
    diamond_manager.setSeasonEndTimestamp(2, get_timestamp(5), {"from":
owner})
    with reverts(error_hex_code('DepositFacet__FundsInPrevSeason()')):
        deposit_facet.deposit(50, {"from": owner})


    with reverts(error_hex_code('DepositFacet__NotEnoughTokenBalance()')):
        deposit_facet.deposit(100, {"from": other})

    # deposit
    amount = 100
    deposit_token.mint(other, amount)
    deposit_token.approve(deposit_facet.address, amount, {"from": other})
    assert diamond_manager.getDepositAmountOfUser(other, 2) == 0
    deposit_facet.deposit(amount, {"from": other})
    assert diamond_manager.getDepositAmountOfUser(other, 2) == 95

def test_deposit_stratosphere(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond = 
deploy_liquid_mining_diamond(owner)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address, 
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet", 
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    stratosphere.setStratosphereMemberBasic(other)
    deposit_token.mint(other, 150)
    amount = 100
    deposit_token.approve(deposit_facet.address, amount, {"from": other})
    assert diamond_manager.getDepositAmountOfUser(other, 1) == 0
    assert diamond_manager.getDepositPointsOfUser(other, 1) == 0
    assert diamond_manager.getTotalDepositAmountOfSeason(1) == 0
    assert diamond_manager.getTotalPointsOfSeason(1) == 0
    deposit_facet.deposit(amount, {"from": other})
    assert diamond_manager.getDepositAmountOfUser(other, 1) == 96
    assert diamond_manager.getDepositPointsOfUser(other, 1) > 0
    assert diamond_manager.getTotalDepositAmountOfSeason(1) == 96
    assert diamond_manager.getTotalPointsOfSeason(1) >  0
```

DiamondManagerFacet:

```
from brownie import (
    reverts,
    DiamondManagerFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    error_hex_code
)

from scripts.deploy import (
    deploy_stratosphere_mock,
    deploy_liquid_mining_diamond
)

def test_set_boost_fee_receivers(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond = 
deploy_liquid_mining_diamond(owner)
    diamond_manager = Contract.from_abi("DiamondManagerFacet", 
diamond.address, DiamondManagerFacet.abi)

    receivers = [get_account(2), get_account(3)]
    proportions = [10, 15]
    with reverts(error_hex_code('DiamondManagerFacet__Not_Owner()')):
        diamond_manager.setBoostFeeReceivers(receivers, proportions, 
{"from": other})

    with reverts(error_hex_code('DiamondManagerFacet__Invalid_Input()')):
        diamond_manager.setBoostFeeReceivers(receivers, [], {"from": 
owner})

    diamond_manager.setBoostFeeReceivers(receivers, proportions, {"from": 
owner})

def test_set_claim_fee_receivers(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond = 
deploy_liquid_mining_diamond(owner)
    diamond_manager = Contract.from_abi("DiamondManagerFacet", 
diamond.address, DiamondManagerFacet.abi)

    receivers = [get_account(2), get_account(3)]
    proportions = [10, 15]
    with reverts(error_hex_code('DiamondManagerFacet__Not_Owner()')):
        diamond_manager.setClaimFeeReceivers(receivers, proportions,
{"from": other})

    with reverts(error_hex_code('DiamondManagerFacet__Invalid_Input()')):
        diamond_manager.setClaimFeeReceivers(receivers, [], {"from":
owner})

    diamond_manager.setClaimFeeReceivers(receivers, proportions, {"from":
owner})

def test_set_deposit_fee_receivers(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [get_account(2), get_account(3)]
    proportions = [10, 15]
    with reverts(error_hex_code('DiamondManagerFacet__Not_Owner()')):
        diamond_manager.setDepositFeeReceivers(receivers, proportions,
{"from": other})

    with reverts(error_hex_code('DiamondManagerFacet__Invalid_Input()')):
        diamond_manager.setDepositFeeReceivers(receivers, [], {"from":
owner})

    diamond_manager.setDepositFeeReceivers(receivers, proportions, {"from":
owner})

def test_set_restake_fee_receivers(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [get_account(2), get_account(3)]
    proportions = [10, 15]
    with reverts(error_hex_code('DiamondManagerFacet__Not_Owner()')):
        diamond_manager.setRestakeFeeReceivers(receivers, proportions,
{"from": other})

    with reverts(error_hex_code('DiamondManagerFacet__Invalid_Input()')):
        diamond_manager.setRestakeFeeReceivers(receivers, [], {"from":
owner})

    diamond_manager.setRestakeFeeReceivers(receivers, proportions, {"from":
owner})

def test_set_stratosphere(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    new_stratos = deploy_stratosphere_mock(owner)
    with reverts(error_hex_code('DiamondManagerFacet__Not_Owner()')):
        diamond_manager.setStratosphereAddress(new_stratos.address,
{"from": other})

    with reverts(error_hex_code('DiamondManagerFacet__Invalid_Address()')):
        diamond_manager.setStratosphereAddress(ZERO_ADDRESS, {"from":
owner})

    assert diamond_manager.getStratosphereAddress() == stratosphere.address
    diamond_manager.setStratosphereAddress(new_stratos.address, {"from":
owner})
    assert diamond_manager.getStratosphereAddress() == new_stratos.address

def test_set_unclock_fee_receivers(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [get_account(2), get_account(3)]
    proportions = [10, 15]
    with reverts(error_hex_code('DiamondManagerFacet__Not_Owner()')):
        diamond_manager.setUnlockFeeReceivers(receivers, proportions,
{"from": other})

    with reverts(error_hex_code('DiamondManagerFacet__Invalid_Input()')):
        diamond_manager.setUnlockFeeReceivers(receivers, [], {"from":
owner})

    diamond_manager.setUnlockFeeReceivers(receivers, proportions, {"from":
owner})
```

FeeCollectorFacet:

```
from brownie import (
    reverts,
    FeeCollectorFacet, ClaimFacet, DepositFacet,
    DiamondManagerFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    error_hex_code,
    get_timestamp,
    evm_increase_time
)

from scripts.deploy import (
    deploy_liquid_mining_diamond
)

def test_collect_boost_fees(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    receiver = get_account(2)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    fee_collector_facet = Contract.from_abi("FeeCollectorFacet",
diamond.address, FeeCollectorFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [receiver]
    proportions = [10]
    diamond_manager.setBoostFeeReceivers(receivers, proportions, {"from":
owner})

    # asserts
    with reverts(error_hex_code("FeeCollectorFacet__Only_Owner()")):
        fee_collector_facet.collectBoostFees({"from": other})

    # mint
    amount = 10000
    boost_fee_token.mint(fee_collector_facet.address, amount)
    boost_fee_token.approve(fee_collector_facet.address, amount)
    assert boost_fee_token.balanceOf(receiver) == 0
    fee_collector_facet.collectBoostFees({"from": owner})
    assert boost_fee_token.balanceOf(receiver) > 0

def test_collect_claim_fees(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    receiver = get_account(2)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    fee_collector_facet = Contract.from_abi("FeeCollectorFacet",
diamond.address, FeeCollectorFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [receiver]
    proportions = [10]
    diamond_manager.setClaimFeeReceivers(receivers, proportions, {"from":
owner})
```

```
    # asserts
    with reverts(error_hex_code("FeeCollectorFacet__Only_Owner()")):
        fee_collector_facet.collectClaimFees({"from": other})

    fee_collector_facet.collectClaimFees({"from": owner})

def test_collect_deposit_fees(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    receiver = get_account(2)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    fee_collector_facet = Contract.from_abi("FeeCollectorFacet",
diamond.address, FeeCollectorFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [receiver]
    proportions = [10]
    diamond_manager.setDepositFeeReceivers(receivers, proportions, {"from":
owner})

    # asserts
    with reverts(error_hex_code("FeeCollectorFacet__Only_Owner()")):
        fee_collector_facet.collectDepositFees({"from": other})

    fee_collector_facet.collectDepositFees({"from": owner})

def test_collect_restake_fees(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    receiver = get_account(2)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    fee_collector_facet = Contract.from_abi("FeeCollectorFacet",
diamond.address, FeeCollectorFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)
```

```
    receivers = [receiver]
    proportions = [10]
    diamond_manager.setRestakeFeeReceivers(receivers, proportions, {"from":
owner})

    # asserts
    with reverts(error_hex_code("FeeCollectorFacet__Only_Owner()")):
        fee_collector_facet.collectRestakeFees({"from": other})

    fee_collector_facet.collectRestakeFees({"from": owner})

def test_collect_unlock_fees(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    receiver = get_account(2)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    fee_collector_facet = Contract.from_abi("FeeCollectorFacet",
diamond.address, FeeCollectorFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    receivers = [receiver]
    proportions = [10]
    diamond_manager.setUnlockFeeReceivers(receivers, proportions, {"from":
owner})

    # asserts
    with reverts(error_hex_code("FeeCollectorFacet__Only_Owner()")):
        fee_collector_facet.collectUnlockFees({"from": other})

    fee_collector_facet.collectUnlockFees({"from": owner})
```

RestakeFacet:

```
from brownie import (
    reverts,
    RestakeFacet, DepositFacet, UnlockFacet,
    DiamondManagerFacet, WithdrawFacet
)
```

```
from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    evm_increase_time,
    error_hex_code
)

from scripts.deploy import (
    deploy_liquid_mining_diamond
)

def test_restake(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    restake_facet = Contract.from_abi("RestakeFacet", diamond.address,
RestakeFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})
    # Asserts
    evm_increase_time(86400) # increase 1 day
    with reverts(error_hex_code('RestakeFacet__InProgressSeason()')):
        restake_facet.restake({"from": owner})

    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 3) # increase 3 days
    with reverts():
        restake_facet.restake({"from": owner})
    diamond_manager.startNewSeason(100, {"from": owner}) # season 2
    assert diamond_manager.getWithdrawRestakeStatus(owner, 1) == False
    restake_facet.restake({"from": owner})
    assert diamond_manager.getWithdrawRestakeStatus(owner, 1) == True

def test_restake_stratosphere(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    restake_facet = Contract.from_abi("RestakeFacet", diamond.address,
RestakeFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    stratosphere.setStratosphereMemberBasic(other)
    amount = 100
    deposit_token.mint(other, amount)
    deposit_token.approve(deposit_facet.address, amount, {"from": other})
    deposit_facet.deposit(amount, {"from": other})
    evm_increase_time(86400) # increase 1 day
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 3) # increase 3 days
    # Asserts
    diamond_manager.startNewSeason(100, {"from": owner}) # season 2
    assert diamond_manager.getWithdrawRestakeStatus(other, 1) == False
    restake_facet.restake({"from": other})
    assert diamond_manager.getWithdrawRestakeStatus(other, 1) == True

def test_restake_with_unlock(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    restake_facet = Contract.from_abi("RestakeFacet", diamond.address,
RestakeFacet.abi)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})
    # Asserts
    evm_increase_time(86400) # increase 1 day
    unlock_facet.unlock(50, {"from": owner})
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 3) # increase 3 days
    with reverts(error_hex_code('RestakeFacet__FundsInPrevSeason()')):
        restake_facet.restake({"from": owner})

def test_restake_with_withdraw(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    restake_facet = Contract.from_abi("RestakeFacet", diamond.address,
RestakeFacet.abi)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    withdraw_facet = Contract.from_abi("WithdrawFacet", diamond.address,
WithdrawFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})
    # Asserts
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 3) # increase 3 days
    withdraw_facet.withdraw({"from": owner})
    with reverts(error_hex_code('RestakeFacet__HasWithdrawnOrRestaked()')):
        restake_facet.restake({"from": owner})
```

UnlockFacet:

```
from brownie import (
    reverts,
    UnlockFacet, DepositFacet,
    DiamondManagerFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    evm_increase_time,
    error_hex_code
)

from scripts.deploy import (
    deploy_liquid_mining_diamond
)

def test_unlock(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)
```

```
    diamond_manager.startNewSeason(100, {"from": owner})
    amount = 200
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})
    # Asserts
    evm_increase_time(86400) # increase 1 day
    with reverts(error_hex_code('UnlockFacet__InvalidAmount()')):
        unlock_facet.unlock(500, {"from": owner})

    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    with reverts(error_hex_code('UnlockFacet__InvalidUnlock()')):
        unlock_facet.unlock(100, {"from": owner})

    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(5), {"from":
owner})
    assert diamond_manager.getUnlockTimestampOfUser(owner, 1) == 0
    assert diamond_manager.getUnlockAmountOfUser(owner, 1) == 0
    assert diamond_manager.getDepositAmountOfUser(owner, 1) == 190
    unlock_facet.unlock(100, {"from": owner})
    assert diamond_manager.getUnlockTimestampOfUser(owner, 1) != 0
    assert diamond_manager.getUnlockAmountOfUser(owner, 1) == 90
    assert diamond_manager.getDepositAmountOfUser(owner, 1) == 90

    with reverts(error_hex_code('UnlockFacet__AlreadyUnlocked()')):
        unlock_facet.unlock(100, {"from": owner})

def test_unlock_stratospher(only_local):
    # arrange
    owner = get_account(0)
    other = get_account(1)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner})
    stratosphere.setStratosphereMemberBasic(other)
    amount = 200
    deposit_token.mint(other, amount)
    deposit_token.approve(deposit_facet.address, amount, {"from": other})
    deposit_facet.deposit(amount, {"from": other})


    assert diamond_manager.getUnlockTimestampOfUser(other, 1) == 0
    assert diamond_manager.getUnlockAmountOfUser(other, 1) == 0
    assert diamond_manager.getDepositAmountOfUser(other, 1) == 191
    unlock_facet.unlock(100, {"from": other})
    assert diamond_manager.getUnlockTimestampOfUser(other, 1) != 0
    assert diamond_manager.getUnlockAmountOfUser(other, 1) == 91
    assert diamond_manager.getDepositAmountOfUser(other, 1) == 91
```

<u>WihtdrawFacet</u>:

```
from brownie import (
    reverts,
    WithdrawFacet, DepositFacet, UnlockFacet,
    DiamondManagerFacet, WithdrawFacet
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_timestamp,
    evm_increase_time,
    error_hex_code
)

from scripts.deploy import (
    deploy_liquid_mining_diamond
)
```

```
def test_withdraw_unlocked(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    withdraw_facet = Contract.from_abi("WithdrawFacet", diamond.address,
WithdrawFacet.abi)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    # Asserts
    with reverts(error_hex_code('WithdrawFacet__InsufficientBalance()')):
        withdraw_facet.withdrawUnlocked({"from": owner})

    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})

    evm_increase_time(86400) # increase 1 day
    unlock_facet.unlock(50, {"from": owner})
    with reverts(error_hex_code('WithdrawFacet__UnlockNotMatured()')):
        withdraw_facet.withdrawUnlocked({"from": owner})
    evm_increase_time(86400 * 5) # increase 5 day

    assert diamond_manager.getUnlockTimestampOfUser(owner, 1) != 50
    assert diamond_manager.getUnlockAmountOfUser(owner, 1) != 0
    withdraw_facet.withdrawUnlocked({"from": owner})
    assert diamond_manager.getUnlockTimestampOfUser(owner, 1) == 0
    assert diamond_manager.getUnlockAmountOfUser(owner, 1) == 0

def test_withdraw(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    withdraw_facet = Contract.from_abi("WithdrawFacet", diamond.address,
WithdrawFacet.abi)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    # Asserts
    with reverts(error_hex_code('WithdrawFacet__UserNotParticipated()')):
        withdraw_facet.withdraw({"from": owner})
    evm_increase_time(86400) # increase 1 day

    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})
    with reverts(error_hex_code('WithdrawFacet__InProgressSeason()')):
        withdraw_facet.withdraw({"from": owner})
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 4) # increase 4 day

    assert diamond_manager.getWithdrawRestakeStatus(owner, 1) == False
    withdraw_facet.withdraw({"from": owner})
    assert diamond_manager.getWithdrawRestakeStatus(owner, 1) == True
    with reverts(error_hex_code('WithdrawFacet__AlreadyWithdrawn()')):
        withdraw_facet.withdraw({"from": owner})


def test_withdraw_all_no_balance(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    withdraw_facet = Contract.from_abi("WithdrawFacet", diamond.address,
WithdrawFacet.abi)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
```

```
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    # Asserts
    with reverts(error_hex_code('WithdrawFacet__UserNotParticipated()')):
        withdraw_facet.withdrawAll({"from": owner})
    evm_increase_time(86400) # increase 1 day

    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})

    with reverts(error_hex_code('WithdrawFacet__InProgressSeason()')):
        withdraw_facet.withdrawAll({"from": owner})
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 4) # increase 4 day

    with reverts(error_hex_code('WithdrawFacet__InsufficientBalance()')):
        withdraw_facet.withdrawAll({"from": owner})

def test_withdraw_all_balance(only_local):
    # arrange
    owner = get_account(0)
    deposit_token, boost_fee_token, reward_token, stratosphere, diamond =
deploy_liquid_mining_diamond(owner)
    withdraw_facet = Contract.from_abi("WithdrawFacet", diamond.address,
WithdrawFacet.abi)
    unlock_facet = Contract.from_abi("UnlockFacet", diamond.address,
UnlockFacet.abi)
    deposit_facet = Contract.from_abi("DepositFacet", diamond.address,
DepositFacet.abi)
    diamond_manager = Contract.from_abi("DiamondManagerFacet",
diamond.address, DiamondManagerFacet.abi)

    diamond_manager.startNewSeason(100, {"from": owner}) # season 1
    # Asserts
    with reverts(error_hex_code('WithdrawFacet__UserNotParticipated()')):
        withdraw_facet.withdrawAll({"from": owner})
    evm_increase_time(86400) # increase 1 day
    amount = 100
    deposit_token.mint(owner, amount)
    deposit_token.approve(deposit_facet.address, amount)
    deposit_facet.deposit(amount, {"from": owner})
    unlock_facet.unlock(50, {"from": owner})
    with reverts(error_hex_code('WithdrawFacet__InProgressSeason()')):
        withdraw_facet.withdrawAll({"from": owner})
    diamond_manager.setSeasonEndTimestamp(1, get_timestamp(2), {"from":
owner})
    evm_increase_time(86400 * 4) # increase 4 day

    assert diamond_manager.getWithdrawRestakeStatus(owner, 1) == False
    withdraw_facet.withdrawAll({"from": owner})
    assert diamond_manager.getWithdrawRestakeStatus(owner, 1) == True

    with reverts(error_hex_code('WithdrawFacet__InsufficientBalance()')):
        withdraw_facet.withdrawAll({"from": owner})
```

5.0 Summary of the audit

We have detected medium-level vulnerabilities that we recommend reviewing alongside the others. Overall, the code is very well-detailed and written in accordance with Solidity standards.