Smart contract security audit Onboard

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During February of 2023, Onboard 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. Onboard 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 Onboard 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 Onboard 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:

<u>Source file:</u>

onboard-token-dev.zip [SHA256] -

5135176924eba2cf1327fc32fba2cca77f6c36d32e82c496011be25813a29252

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.	PASSED
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 [0]

No medium severity issues found.

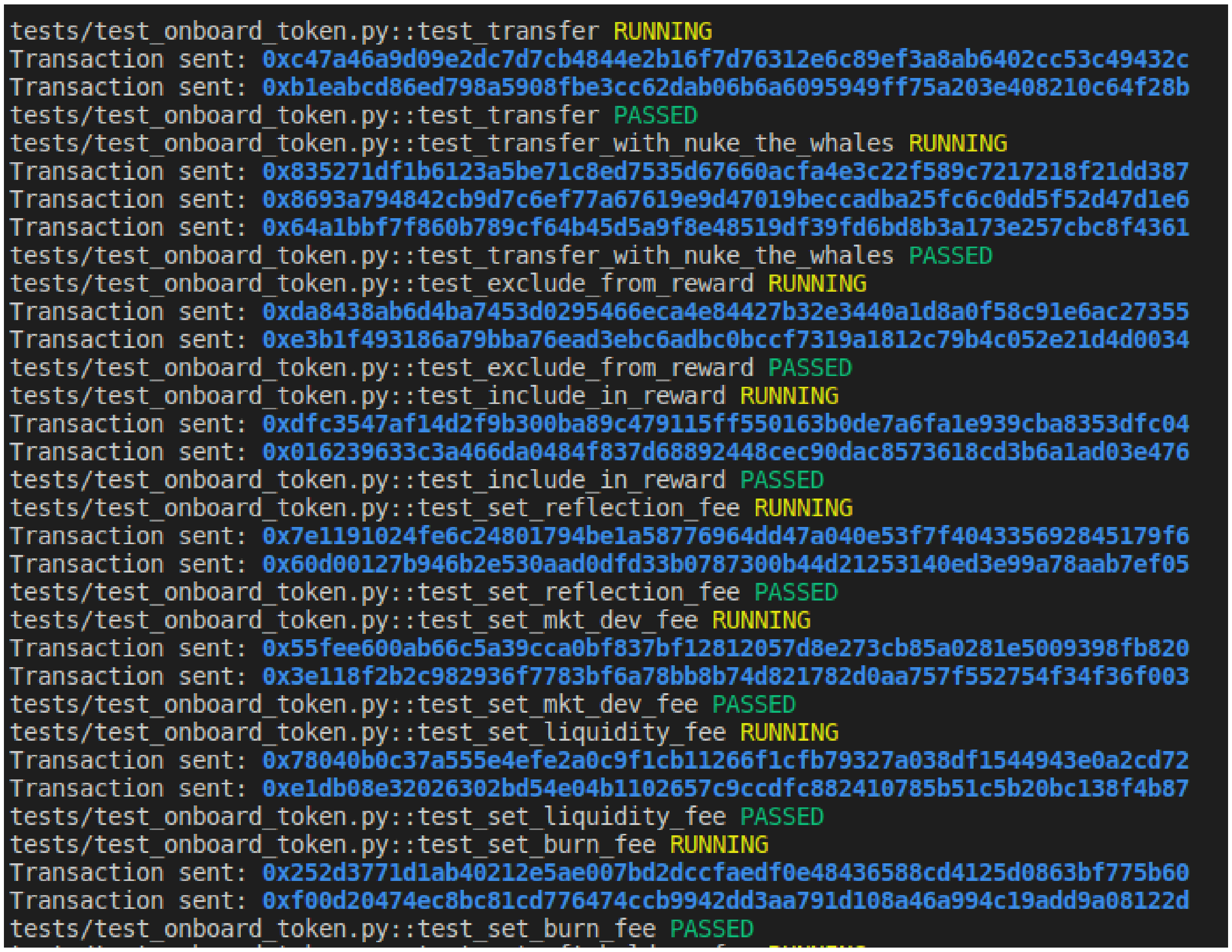
3.3 Low severity issues [0]

No low severity issues found.

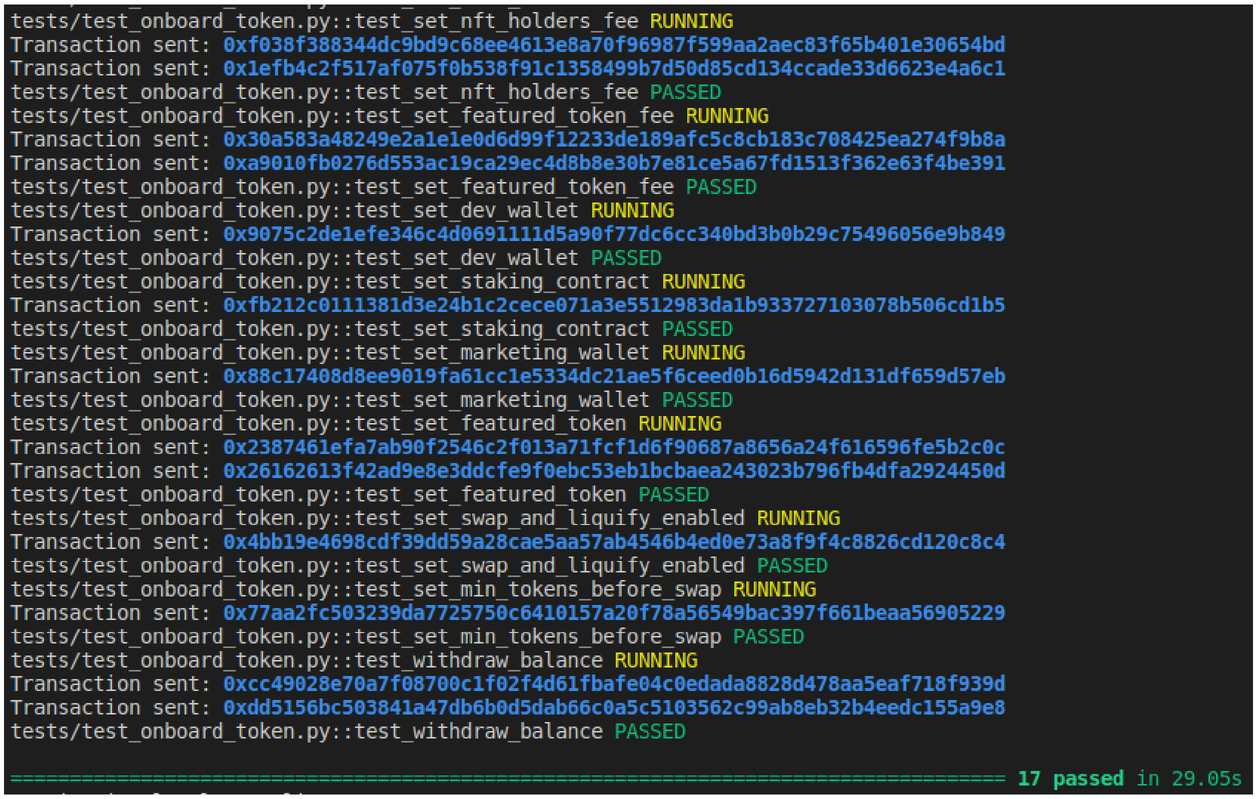
4.0 Testing coverage - python

During the testing phase, custom use cases were written to cover all the logic of contracts in python language. *<u>*Check "5 Annexes" to see the testing code.</u>*

<u>Onboard tests</u>



```
tests/test_onboard_token.py::test_transfer RUNNING
Transaction sent: 0xc47a46a9d09e2dc7d7cb4844e2b16f7d76312e6c89ef3a8ab6402cc53c49432c
Transaction sent: 0xb1eabcd86ed798a5908fbe3cc62dab06b6a6095949ff75a203e408210c64f28b
tests/test_onboard_token.py::test_transfer PASSED
tests/test_onboard_token.py::test_transfer_with_nuke_the_whales RUNNING
Transaction sent: 0x835271df1b6123a5be71c8ed7535d67660acfa4e3c22f589c7217218f21dd387
Transaction sent: 0x8693a794842cb9d7c6ef77a67619e9d47019beccadba25fc6c0dd5f52d47d1e6
Transaction sent: 0x64a1bbf7f860b789cf64b45d5a9f8e48519df39fd6bd8b3a173e257cbc8f4361
tests/test_onboard_token.py::test_transfer_with_nuke_the_whales PASSED
tests/test_onboard_token.py::test_exclude_from_reward RUNNING
Transaction sent: 0xda8438ab6d4ba7453d0295466eca4e84427b32e3440a1d8a0f58c91e6ac27355
Transaction sent: 0xe3b1f493186a79bba76ead3ebc6adbc0bccf7319a1812c79b4c052e21d4d0034
tests/test_onboard_token.py::test_exclude_from_reward PASSED
tests/test_onboard_token.py::test_include_in_reward RUNNING
Transaction sent: 0xdfc3547af14d2f9b300ba89c479115ff550163b0de7a6fa1e939cba8353dfc04
Transaction sent: 0x016239633c3a466da0484f837d68892448cec90dac8573618cd3b6a1ad03e476
tests/test_onboard_token.py::test_include_in_reward PASSED
tests/test_onboard_token.py::test_set_reflection_fee RUNNING
Transaction sent: 0x7e1191024fe6c24801794be1a58776964dd47a040e53f7f404335692845179f6
Transaction sent: 0x60d00127b946b2e530aad0dfd33b0787300b44d21253140ed3e99a78aab7ef05
tests/test_onboard_token.py::test_set_reflection_fee PASSED
tests/test_onboard_token.py::test_set_mkt_dev_fee RUNNING
Transaction sent: 0x55fee600ab66c5a39cca0bf837bf12812057d8e273cb85a0281e5009398fb820
Transaction sent: 0x3e118f2b2c982936f7783bf6a78bb8b74d821782d0aa757f552754f34f36f003
tests/test_onboard_token.py::test_set_mkt_dev_fee PASSED
tests/test_onboard_token.py::test_set_liquidity_fee RUNNING
Transaction sent: 0x78040b0c37a555e4efe2a0c9f1cb11266f1cfb79327a038df1544943e0a2cd72
Transaction sent: 0xe1db08e32026302bd54e04b1102657c9ccdfc882410785b51c5b20bc138f4b87
tests/test_onboard_token.py::test_set_liquidity_fee PASSED
tests/test_onboard_token.py::test_set_burn_fee RUNNING
Transaction sent: 0x252d3771d1ab40212e5ae007bd2dccfaedf0e48436588cd4125d0863bf775b60
Transaction sent: 0xf00d20474ec8bc81cd776474ccb9942dd3aa791d108a46a994c19add9a08122d
tests/test_onboard_token.py::test_set_burn_fee PASSED
```



```
tests/test_onboard_token.py::test_set_nft_holders_fee RUNNING
Transaction sent: 0xf038f388344dc9bd9c68ee4613e8a70f96987f599aa2aec83f65b401e30654bd
Transaction sent: 0x1efb4c2f517af075f0b538f91c1358499b7d50d85cd134ccade33d6623e4a6c1
tests/test_onboard_token.py::test_set_nft_holders_fee PASSED
tests/test_onboard_token.py::test_set_featured_token_fee RUNNING
Transaction sent: 0x30a583a48249e2a1e1e0d6d99f12233de189afc5c8cb183c708425ea274f9b8a
Transaction sent: 0xa9010fb0276d553ac19ca29ec4d8b8e30b7e81ce5a67fd1513f362e63f4be391
tests/test_onboard_token.py::test_set_featured_token_fee PASSED
tests/test_onboard_token.py::test_set_dev_wallet RUNNING
Transaction sent: 0x9075c2de1efe346c4d0691111d5a90f77dc6cc340bd3b0b29c75496056e9b849
tests/test_onboard_token.py::test_set_dev_wallet PASSED
tests/test_onboard_token.py::test_set_staking_contract RUNNING
Transaction sent: 0xfb212c0111381d3e24b1c2cece071a3e5512983da1b933727103078b506cd1b5
tests/test_onboard_token.py::test_set_staking_contract PASSED
tests/test_onboard_token.py::test_set_marketing_wallet RUNNING
Transaction sent: 0x88c17408d8ee9019fa61cc1e5334dc21ae5f6ceed0b16d5942d131df659d57eb
tests/test_onboard_token.py::test_set_marketing_wallet PASSED
tests/test_onboard_token.py::test_set_featured_token RUNNING
Transaction sent: 0x2387461efa7ab90f2546c2f013a71fcf1d6f90687a8656a24f616596fe5b2c0c
Transaction sent: 0x26162613f42ad9e8e3ddcfe9f0ebc53eb1bcbaea243023b796fb4dfa2924450d
tests/test_onboard_token.py::test_set_featured_token PASSED
tests/test_onboard_token.py::test_set_swap_and_liquify_enabled RUNNING
Transaction sent: 0x4bb19e4698cdf39dd59a28cae5aa57ab4546b4ed0e73a8f9f4c8826cd120c8c4
tests/test_onboard_token.py::test_set_swap_and_liquify_enabled PASSED
tests/test_onboard_token.py::test_set_min_tokens_before_swap RUNNING
Transaction sent: 0x77aa2fc503239da7725750c6410157a20f78a56549bac397f661beaa56905229
tests/test_onboard_token.py::test_set_min_tokens_before_swap PASSED
tests/test_onboard_token.py::test_withdraw_balance RUNNING
Transaction sent: 0xcc49028e70a7f08700c1f02f4d61fbafe04c0edada8828d478aa5eaf718f939d
Transaction sent: 0xdd5156bc503841a47db6b0d5dab66c0a5c5103562c99ab8eb32b4eedc155a9e8
tests/test_onboard_token.py::test_withdraw_balance PASSED

==================================================================================== 17 passed in 29.05s
```

5.0 Annexes

Onboard testing code:

```
from brownie import (
    reverts,
    UniswapV2Factory,
    UniswapV2Router02,
    OnboardToken
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
)

from scripts.deploy import (
    deploy_test_weth,
    deploy_test_erc20,
    deploy_onboard_token
)

def test_transfer(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    onboard = deploy_onboard_token(owner)

    # assert
    with reverts("ERC20: transfer to the zero address"):
        onboard.transfer(ZERO_ADDRESS, 100, {"from": owner})

    with reverts("Transfer amount must be greater than zero"):
        onboard.transfer(onboard.address, 0, {"from": owner})

    prevBalance = onboard.balanceOf(owner)
    value = 200

    tx = onboard.transfer(onboard.address, value, {"from": owner})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"]["from"] == owner
    assert tx.events["Transfer"]["to"] == onboard.address
    assert tx.events["Transfer"]["value"] == value
    assert onboard.balanceOf(onboard.address) == value
    assert onboard.balanceOf(owner) == prevBalance - value

    tx1 = onboard.transfer(other, value, {"from": owner})
    assert tx1.events["Transfer"] is not None
    assert tx1.events["Transfer"]["from"] == owner
    assert tx1.events["Transfer"]["to"] == other
    assert tx1.events["Transfer"]["value"] == value
    assert onboard.balanceOf(other) == value
    assert onboard.balanceOf(owner) == prevBalance - value - value

    # force liquify
    onboard.setSwapAndLiquifyEnabled(True, {"from": owner})
    onboard.setMinTokensBeforeSwap(50, {"from": owner})
    onboard.transfer(onboard.address, 10, {"from": other})
    assert onboard.balanceOf(onboard.address) == 10
    assert onboard.balanceOf(other) == value - 10

    tx3 = onboard.transfer(extra, 10, {"from": other})
    assert tx3.events["Transfer"] is not None
    assert tx3.events["Transfer"]["from"] == other
    assert tx3.events["Transfer"]["to"] == extra
    assert tx3.events["Transfer"]["value"] == 10
    assert onboard.balanceOf(extra) == 10
    assert onboard.balanceOf(other) == value - 10 - 10

def test_transfer_with_nuke_the_whales(only_local):
     #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    onboard = deploy_onboard_token(owner)

    onboard.startNukingTheWhales({"from": owner})
    # Transfer some tokens
    value = onboard.balanceOf(owner) * 0.2
    tx = onboard.transfer(other, value, {"from": owner})
    # assert
    with reverts("Transfer amount exceeds the 0.1% of the supply."):
        onboard.transfer(extra, value, {"from": other})
    uniswap_pair_address = onboard.uniswapV2Pair()
    with reverts("For your protection, max sell is 20% if you hold 0.5% or
more of supply."):
        onboard.transfer(uniswap_pair_address, value, {"from": owner})
    new_value = onboard.balanceOf(owner) * 0.01
    with reverts("You must wait a full 24 hours before you may sell
again."):
        onboard.transfer(uniswap_pair_address, new_value, {"from": owner})

def test_exclude_from_reward(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    external = get_account(2)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.excludeFromReward(other, {"from": external})

    onboard.excludeFromReward(other, {"from": owner})
    with reverts("Account is already excluded"):
        onboard.excludeFromReward(other, {"from": owner})

def test_include_in_reward(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    external = get_account(2)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.includeInReward(external, {"from": other})
    with reverts("Account is already included"):
        onboard.includeInReward(external, {"from": owner})
    onboard.excludeFromReward(external, {"from": owner})
    onboard.includeInReward(external, {"from": owner})
```

```
def test_set_reflection_fee(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setReflectionFee(5, {"from": other})
    with reverts("Onboard token: fee exceeds max permitted"):
        onboard.setReflectionFee(2500, {"from": owner})

    previous = 1
    assert onboard.fees()['reflectionFee'] == previous
    onboard.setReflectionFee(2, {"from": owner})
    assert onboard.fees()['reflectionFee'] == 2

def test_set_mkt_dev_fee(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setMktDevFee(5, {"from": other})
    with reverts("Onboard token: fee exceeds max permitted"):
        onboard.setMktDevFee(2500, {"from": owner})

    previous = 3
    assert onboard.fees()['mktDevFee'] == previous
    onboard.setMktDevFee(2, {"from": owner})
    assert onboard.fees()['mktDevFee'] == 2

def test_set_liquidity_fee(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setLiquidityFee(5, {"from": other})
    with reverts("Onboard token: fee exceeds max permitted"):
        onboard.setLiquidityFee(2500, {"from": owner})

    previous = 1
    assert onboard.fees()['liquidityFee'] == previous
    onboard.setLiquidityFee(2, {"from": owner})
    assert onboard.fees()['liquidityFee'] == 2

def test_set_burn_fee(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setBurnFee(5, {"from": other})
    with reverts("Onboard token: fee exceeds max permitted"):
        onboard.setBurnFee(2500, {"from": owner})

    previous = 1
    assert onboard.fees()['burnFee'] == previous
    onboard.setBurnFee(2, {"from": owner})
    assert onboard.fees()['burnFee'] == 2

def test_set_nft_holders_fee(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setNftHoldersFee(5, {"from": other})
    with reverts("Onboard token: fee exceeds max permitted"):
        onboard.setNftHoldersFee(2500, {"from": owner})

    previous = 2
    assert onboard.fees()['nftHoldersFee'] == previous
    onboard.setNftHoldersFee(3, {"from": owner})
    assert onboard.fees()['nftHoldersFee'] == 3
```

```
def test_set_featured_token_fee(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setFeaturedTokenFee(5, {"from": other})
    with reverts("Onboard token: fee exceeds max permitted"):
        onboard.setFeaturedTokenFee(2500, {"from": owner})

    previous = 2
    assert onboard.fees()['featTokenFee'] == previous
    onboard.setFeaturedTokenFee(3, {"from": owner})
    assert onboard.fees()['featTokenFee'] == 3

def test_set_dev_wallet(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setDevWallet(other, {"from": extra})
    assert onboard._devWallet() == owner
    onboard.setDevWallet(other, {"from": owner})
    assert onboard._devWallet() == other

def test_set_staking_contract(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setStakingContract(other, {"from": extra})
    assert onboard._stakingContract() == owner
    onboard.setStakingContract(other, {"from": owner})
    assert onboard._stakingContract() == other

def test_set_marketing_wallet(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setMarketingWallet(other, {"from": extra})
    assert onboard._marketingWallet() == owner
    onboard.setMarketingWallet(other, {"from": owner})
    assert onboard._marketingWallet() == other

def test_set_featured_token(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    # Deploy Mock ERC20
    erc20 = deploy_test_erc20(owner)
    # Deploy WETH
    weth = deploy_test_weth(owner)
    # Deploy UniswapV2Factory
    factory = UniswapV2Factory.deploy(owner, {"from": owner})
    factory.createPair(erc20.address, weth.address)
    # Deploy Router
    v2_router = UniswapV2Router02.deploy(factory.address, weth.address,
{"from": owner})

    # Deploy OnboardToken
    onboard = OnboardToken.deploy(
        erc20.address,
        v2_router.address,
        {"from": owner})

    new_erc20 = deploy_test_erc20(owner)

    #assert
    with reverts():
        onboard.setFeaturedToken(new_erc20.address, {"from": other})
    with reverts("Featured token must have a BNB pair"):
        onboard.setFeaturedToken(new_erc20.address, {"from": owner})
    assert onboard._featuredToken() == erc20.address
    factory.createPair(new_erc20.address, weth.address)
    onboard.setFeaturedToken(new_erc20.address, {"from": owner})
    assert onboard._featuredToken() == new_erc20.address

def test_set_swap_and_liquify_enabled(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    #assert
    with reverts():
        onboard.setSwapAndLiquifyEnabled(True, {"from": other})
    assert onboard.swapAndLiquifyEnabled() == False
    onboard.setSwapAndLiquifyEnabled(True, {"from": owner})
    assert onboard.swapAndLiquifyEnabled() == True

def test_set_min_tokens_before_swap(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    onboard = deploy_onboard_token(owner)

    new_min_token = 10000000 * 10**2
    #assert
    with reverts():
        onboard.setMinTokensBeforeSwap(new_min_token, {"from": other})
    assert onboard.getMinTokensBeforSwap() == 30000000 * 10** 2
    onboard.setMinTokensBeforeSwap(new_min_token, {"from": owner})
    assert onboard.getMinTokensBeforSwap() == new_min_token

def test_withdraw_balance(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    onboard = deploy_onboard_token(owner)
    # assert
    with reverts():
        onboard.withdrawBalance(extra, {"from": other})
    with reverts("Nothing to withdraw"):
        onboard.withdrawBalance(extra, {"from": owner})

    value = 200
    # some random transfers
    onboard.transfer(onboard.address, value, {"from": owner})
    onboard.transfer(other, value, {"from": owner})

    # force liquify
    onboard.setSwapAndLiquifyEnabled(True, {"from": owner})
    onboard.setMinTokensBeforeSwap(50, {"from": owner})
    onboard.transfer(onboard.address, 10, {"from": other})

    # withdraw
    onboard.withdrawBalance(extra, {"from": owner})
```

6.0 Summary of the audit

No high/medium criticality vulnerabilities found at the onboard contracts.