Smart contract security audit

Artify

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During March of 2023, Artify 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. Artify 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 Artify 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 Artify 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 file:

https://etherscan.io/token/0xa41161af8d4ee421ba5fed4328f2b12034796a21#code

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 [2]

1. Warning Against Security Risks of Low-Level Calls in Solidity

Function: swapBack()

Performing low-level calls such as (success,) = address(devAddress).call{value: ethForDev}("") and (success,) = address(operationsAddress).call{value: address(this).balance}("") is not advised as it can potentially pose a security risk.

Function: withdrawStuckETH()

Performing low-level calls that involve (success,) = address(operationsAddress).call{value: address(this).balance}("") is not recommended due to potential security risks.

Fix: To address this, it is advised to utilize secure alternatives such as TransferHelper.safeTransferETH().

2. Preventing Contract Malfunction with a 'Require' Statement to Check for Zero Address Pair

Issue: The presence of a pair zero address can result in incorrect behavior of the contract.

Functions affected: setAutomatedMarketMakerPair() & transferForeignToken()

Resolution: To prevent this issue, include a require statement to verify that the pair is not equal to address(0).

Fix: setAutomatedMarketMakerPair() : pair != address(0) & transferForeignToken() : IERC20(_token).balanceOf(address(this)) > 0.

3.4 Informational issues [1]

1. Optimizing Gas Usage by Reducing Airdrop Token Transfer Amounts

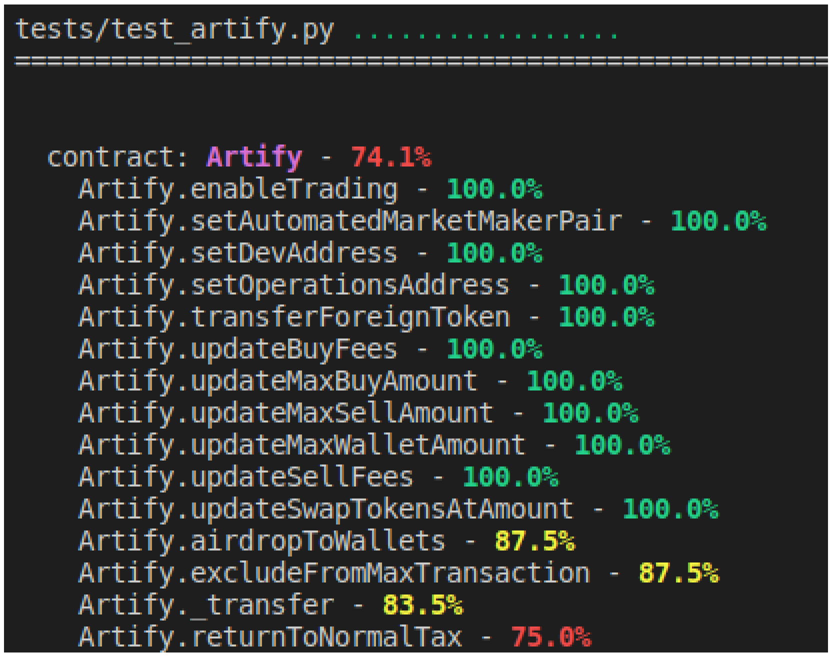
Issue: To avoid gas-related problems, it is advisable to reduce the transfer amount of airdrop tokens.

Resolution: Decrease the specified require of wallets.length < 600 to a lower value.

4.0 Testing coverage - python

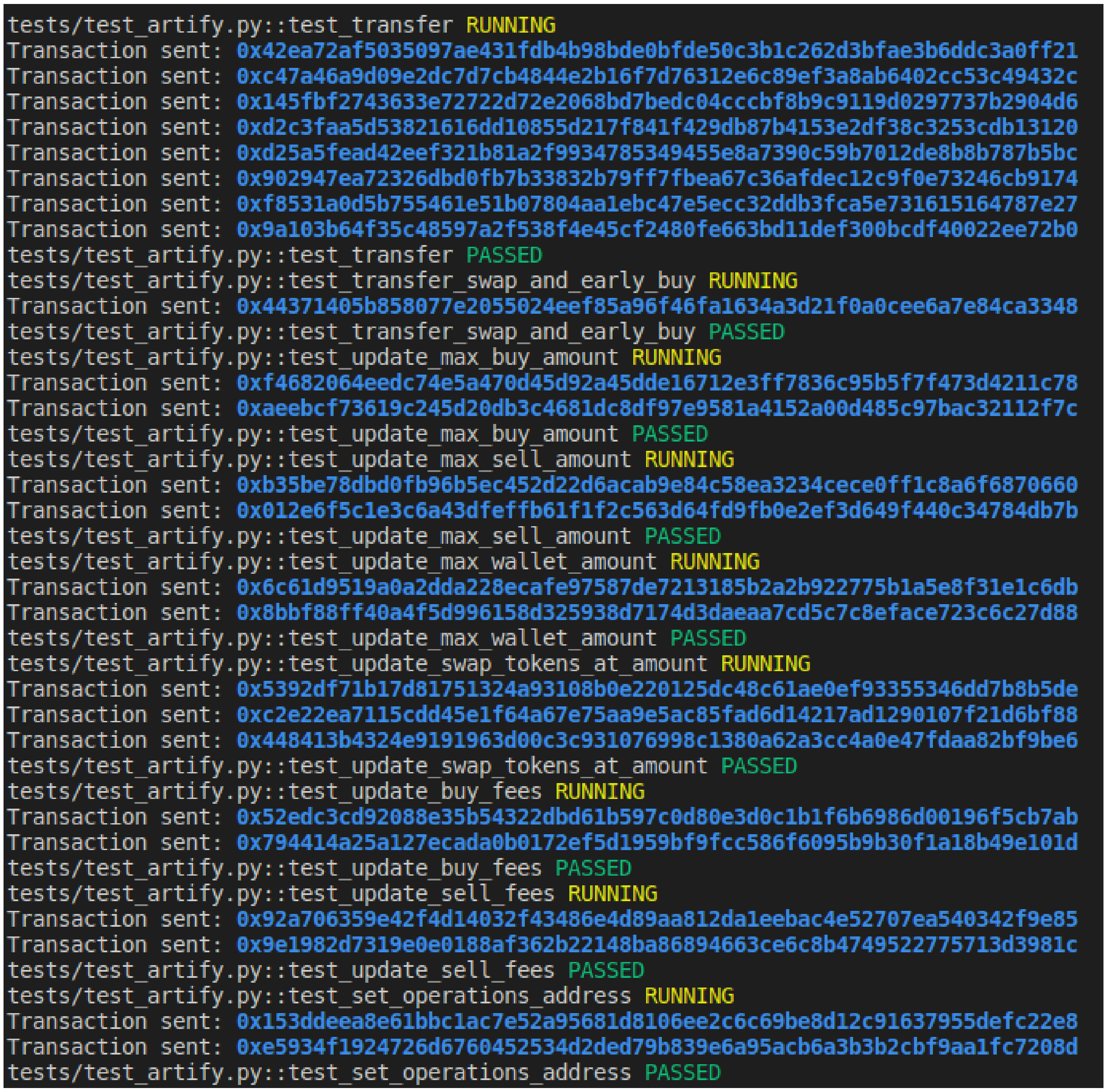
During the testing phase, custom use cases were written to cover all the logic of contracts in python language. *Check "5 Annexes" to see the testing code.*

Artify tests

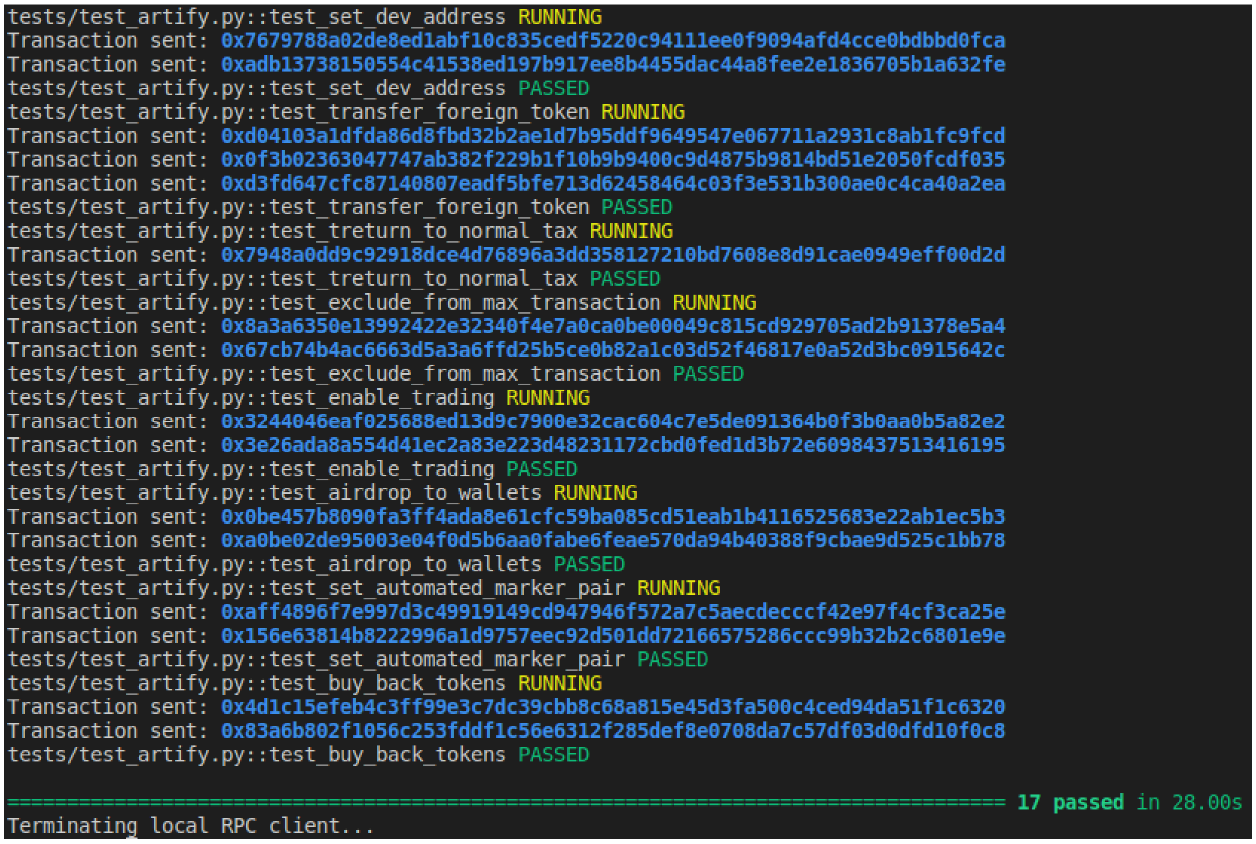


```
tests/test_artify.py .................
=====================================================

  contract: Artify - 74.1%
    Artify.enableTrading - 100.0%
    Artify.setAutomatedMarketMakerPair - 100.0%
    Artify.setDevAddress - 100.0%
    Artify.setOperationsAddress - 100.0%
    Artify.transferForeignToken - 100.0%
    Artify.updateBuyFees - 100.0%
    Artify.updateMaxBuyAmount - 100.0%
    Artify.updateMaxSellAmount - 100.0%
    Artify.updateMaxWalletAmount - 100.0%
    Artify.updateSellFees - 100.0%
    Artify.updateSwapTokensAtAmount - 100.0%
    Artify.airdropToWallets - 87.5%
    Artify.excludeFromMaxTransaction - 87.5%
    Artify._transfer - 83.5%
    Artify.returnToNormalTax - 75.0%
```



```
tests/test_artify.py::test_transfer RUNNING
Transaction sent: 0x42ea72af5035097ae431fdb4b98bde0bfde50c3b1c262d3bfae3b6ddc3a0ff21
Transaction sent: 0xc47a46a9d09e2dc7d7cb4844e2b16f7d76312e6c89ef3a8ab6402cc53c49432c
Transaction sent: 0x145fbf2743633e72722d72e2068bd7bedc04cccbf8b9c9119d0297737b2904d6
Transaction sent: 0xd2c3faa5d53821616dd10855d217f841f429db87b4153e2df38c3253cdb13120
Transaction sent: 0xd25a5fead42eef321b81a2f9934785349455e8a7390c59b7012de8b8b787b5bc
Transaction sent: 0x902947ea72326dbd0fb7b33832b79ff7fbea67c36afdec12c9f0e73246cb9174
Transaction sent: 0xf8531a0d5b755461e51b07804aa1ebc47e5ecc32ddb3fca5e731615164787e27
Transaction sent: 0x9a103b64f35c48597a2f538f4e45cf2480fe663bd11def300bcdf40022ee72b0
tests/test_artify.py::test_transfer PASSED
tests/test_artify.py::test_transfer_swap_and_early_buy RUNNING
Transaction sent: 0x44371405b858077e2055024eef85a96f46fa1634a3d21f0a0cee6a7e84ca3348
tests/test_artify.py::test_transfer_swap_and_early_buy PASSED
tests/test_artify.py::test_update_max_buy_amount RUNNING
Transaction sent: 0xf4682064eedc74e5a470d45d92a45dde16712e3ff7836c95b5f7f473d4211c78
Transaction sent: 0xaeebcf73619c245d20db3c4681dc8df97e9581a4152a00d485c97bac32112f7c
tests/test_artify.py::test_update_max_buy_amount PASSED
tests/test_artify.py::test_update_max_sell_amount RUNNING
Transaction sent: 0xb35be78dbd0fb96b5ec452d22d6acab9e84c58ea3234cece0ff1c8a6f6870660
Transaction sent: 0x012e6f5c1e3c6a43dfeffb61f1f2c563d64fd9fb0e2ef3d649f440c34784db7b
tests/test_artify.py::test_update_max_sell_amount PASSED
tests/test_artify.py::test_update_max_wallet_amount RUNNING
Transaction sent: 0x6c61d9519a0a2dda228ecafe97587de7213185b2a2b922775b1a5e8f31e1c6db
Transaction sent: 0x8bbf88ff40a4f5d996158d325938d7174d3daeaa7cd5c7c8eface723c6c27d88
tests/test_artify.py::test_update_max_wallet_amount PASSED
tests/test_artify.py::test_update_swap_tokens_at_amount RUNNING
Transaction sent: 0x5392df71b17d81751324a93108b0e220125dc48c61ae0ef93355346dd7b8b5de
Transaction sent: 0xc2e22ea7115cdd45e1f64a67e75aa9e5ac85fad6d14217ad1290107f21d6bf88
Transaction sent: 0x448413b4324e9191963d00c3c931076998c1380a62a3cc4a0e47fdaa82bf9be6
tests/test_artify.py::test_update_swap_tokens_at_amount PASSED
tests/test_artify.py::test_update_buy_fees RUNNING
Transaction sent: 0x52edc3cd92088e35b54322dbd61b597c0d80e3d0c1b1f6b6986d00196f5cb7ab
Transaction sent: 0x794414a25a127ecada0b0172ef5d1959bf9fcc586f6095b9b30f1a18b49e101d
tests/test_artify.py::test_update_buy_fees PASSED
tests/test_artify.py::test_update_sell_fees RUNNING
Transaction sent: 0x92a706359e42f4d14032f43486e4d89aa812da1eebac4e52707ea540342f9e85
Transaction sent: 0x9e1982d7319e0e0188af362b22148ba86894663ce6c8b4749522775713d3981c
tests/test_artify.py::test_update_sell_fees PASSED
tests/test_artify.py::test_set_operations_address RUNNING
Transaction sent: 0x153ddeea8e61bbc1ac7e52a95681d8106ee2c6c69be8d12c91637955defc22e8
Transaction sent: 0xe5934f1924726d6760452534d2ded79b839e6a95acb6a3b3b2cbf9aa1fc7208d
tests/test_artify.py::test_set_operations_address PASSED
```



```
tests/test_artify.py::test_set_dev_address RUNNING
Transaction sent: 0x7679788a02de8ed1abf10c835cedf5220c94111ee0f9094afd4cce0bdbbd0fca
Transaction sent: 0xadb13738150554c41538ed197b917ee8b4455dac44a8fee2e1836705b1a632fe
tests/test_artify.py::test_set_dev_address PASSED
tests/test_artify.py::test_transfer_foreign_token RUNNING
Transaction sent: 0xd04103a1dfda86d8fbd32b2ae1d7b95ddf9649547e067711a2931c8ab1fc9fcd
Transaction sent: 0x0f3b02363047747ab382f229b1f10b9b9400c9d4875b9814bd51e2050fcdf035
Transaction sent: 0xd3fd647cfc87140807eadf5bfe713d62458464c03f3e531b300ae0c4ca40a2ea
tests/test_artify.py::test_transfer_foreign_token PASSED
tests/test_artify.py::test_treturn_to_normal_tax RUNNING
Transaction sent: 0x7948a0dd9c92918dce4d76896a3dd358127210bd7608e8d91cae0949eff00d2d
tests/test_artify.py::test_treturn_to_normal_tax PASSED
tests/test_artify.py::test_exclude_from_max_transaction RUNNING
Transaction sent: 0x8a3a6350e13992422e32340f4e7a0ca0be00049c815cd929705ad2b91378e5a4
Transaction sent: 0x67cb74b4ac6663d5a3a6ffd25b5ce0b82a1c03d52f46817e0a52d3bc0915642c
tests/test_artify.py::test_exclude_from_max_transaction PASSED
tests/test_artify.py::test_enable_trading RUNNING
Transaction sent: 0x3244046eaf025688ed13d9c7900e32cac604c7e5de091364b0f3b0aa0b5a82e2
Transaction sent: 0x3e26ada8a554d41ec2a83e223d48231172cbd0fed1d3b72e6098437513416195
tests/test_artify.py::test_enable_trading PASSED
tests/test_artify.py::test_airdrop_to_wallets RUNNING
Transaction sent: 0x0be457b8090fa3ff4ada8e61cfc59ba085cd51eab1b4116525683e22ab1ec5b3
Transaction sent: 0xa0be02de95003e04f0d5b6aa0fabe6feae570da94b40388f9cbae9d525c1bb78
tests/test_artify.py::test_airdrop_to_wallets PASSED
tests/test_artify.py::test_set_automated_marker_pair RUNNING
Transaction sent: 0xaff4896f7e997d3c49919149cd947946f572a7c5aecdecccf42e97f4cf3ca25e
Transaction sent: 0x156e63814b8222996a1d9757eec92d501dd72166575286ccc99b32b2c6801e9e
tests/test_artify.py::test_set_automated_marker_pair PASSED
tests/test_artify.py::test_buy_back_tokens RUNNING
Transaction sent: 0x4d1c15efeb4c3ff99e3c7dc39cbb8c68a815e45d3fa500c4ced94da51f1c6320
Transaction sent: 0x83a6b802f1056c253fddf1c56e6312f285def8e0708da7c57df03d0dfd10f0c8
tests/test_artify.py::test_buy_back_tokens PASSED

===================================================================================== 17 passed in 28.00s
Terminating local RPC client...
```

5.0 Annexes

```
Artify testing code:
from brownie import (
    reverts,
    UniswapV2Factory,
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
)

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

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

    #assert
    with reverts("ERC20: transfer from the zero address"):
        artify.transfer(owner, 100, {"from": ZERO_ADDRESS})
    with reverts("ERC20: transfer to the zero address"):
        artify.transfer(ZERO_ADDRESS, 100, {"from": owner})
    with reverts("amount must be greater than 0"):
        artify.transfer(extra, 0, {"from": other})
    with reverts("Trading is not active."):
        artify.transfer(extra, 50, {"from": other})

    # transfer without fees
    prevBalance = artify.balanceOf(owner)
    value = 1000
    tx = artify.transfer(other, value, {"from": owner})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"]["from"] == owner
    assert tx.events["Transfer"]["to"] == other
    assert tx.events["Transfer"]["value"] == value
    assert artify.balanceOf(other) == value
    assert artify.balanceOf(owner) == prevBalance - value
    # transfer sell
    artify.enableTrading(1, {"from": owner})
    tx2 = artify.transfer(artify.lpPair(), 100, {"from": other})
    # random transfer to increase block number
    artify.transfer(other, value, {"from": owner})
    # transfer buy
    tx3 = artify.transfer(extra, 50, {"from": artify.lpPair()})
    with reverts("_transfer:: Transfer Delay enabled.  Try again later."):
        artify.transfer(other, 50, {"from": artify.lpPair()})
    # allow transfer removing delay
    artify.disableTransferDelay({"from":owner})
    with reverts("Buy transfer amount exceeds the max buy."):
        artify.transfer(other, artify.totalSupply(), {"from":
artify.lpPair()})
    with reverts("Cannot Exceed max wallet"):
        artify.transfer(other, (artify.totalSupply() / 100) +1, {"from":
artify.lpPair()})
    artify.transfer(other, 50, {"from": artify.lpPair()})

    with reverts("Sell transfer amount exceeds the max sell."):
        artify.transfer(artify.lpPair(), artify.totalSupply(), {"from":
other})

def test_transfer_swap_and_early_buy(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    artify = deploy_artify(owner)

    #assert
    # transfer without fees
    value = 1000

    artify.transfer(other, value, {"from": owner})
    # transfer sell and buy in short time
    artify.enableTrading(5, {"from": owner})
    artify.transfer(artify.lpPair(), 100, {"from": other})
    tx = artify.transfer(extra, 50, {"from": artify.lpPair()})
    assert tx.events["CaughtEarlyBuyer"] is not None
    assert tx.events["CaughtEarlyBuyer"][0]['sniper'] == extra
    # test swap and liquify with revert (insufficient liquidity) due
contract constructor restrictions.
    artify.disableTransferDelay({"from": owner})
    artify.transfer(artify.address, (artify.totalSupply() / 10000) + 1000,
{"from": owner})
    with reverts():
        artify.transfer(extra, value, {"from": other})

def test_update_max_buy_amount(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    invalid_amount = 1000000000
    #assert
    with reverts():
        artify.updateMaxBuyAmount(invalid_amount, {"from": other})
    with reverts("Cannot set max buy amount lower than 0.2%"):
        artify.updateMaxBuyAmount(invalid_amount, {"from": owner})

    valid_amount = 10000000000
    assert artify.maxBuyAmount() == artify.totalSupply() * 1 / 100
    artify.updateMaxBuyAmount(valid_amount + 1, {"from": owner})
    assert artify.maxBuyAmount() == (valid_amount + 1) * (10**18)

def test_update_max_sell_amount(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    invalid_amount = 1000000000
    #assert
    with reverts():
        artify.updateMaxSellAmount(invalid_amount, {"from": other})
    with reverts("Cannot set max sell amount lower than 0.2%"):
        artify.updateMaxSellAmount(invalid_amount, {"from": owner})

    valid_amount = 10000000000
    assert artify.maxSellAmount() == artify.totalSupply() * 1 / 100
    artify.updateMaxSellAmount(valid_amount + 1, {"from": owner})
    assert artify.maxSellAmount() == (valid_amount + 1) * (10**18)

def test_update_max_wallet_amount(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    invalid_amount = 1000000000
    #assert
    with reverts():
        artify.updateMaxWalletAmount(invalid_amount, {"from": other})
    with reverts("Cannot set max wallet amount lower than 0.3%"):
        artify.updateMaxWalletAmount(invalid_amount, {"from": owner})

    valid_amount = 15000000000
    assert artify.maxWalletAmount() == artify.totalSupply() * 1 / 100
    artify.updateMaxWalletAmount(valid_amount + 1, {"from": owner})
    assert artify.maxWalletAmount() == (valid_amount + 1) * (10**18)

def test_update_swap_tokens_at_amount(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    invalid_amount = 1000000000
    invalid_amount_2 = 1000000000 * 10e18
    #assert
    with reverts():
        artify.updateSwapTokensAtAmount(invalid_amount, {"from": other})
    with reverts("Swap amount cannot be lower than 0.001% total supply."):
        artify.updateSwapTokensAtAmount(invalid_amount, {"from": owner})
    with reverts("Swap amount cannot be higher than 0.1% total supply."):
        artify.updateSwapTokensAtAmount(invalid_amount_2, {"from": owner})
    valid_amount = 6000000000000000000000000000
    assert artify.swapTokensAtAmount() == artify.totalSupply() * 1 / 10000
    artify.updateSwapTokensAtAmount(valid_amount, {"from": owner})
    assert artify.swapTokensAtAmount() == valid_amount

def test_update_buy_fees(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.updateBuyFees(1, 2, 2, 4, {"from": other})
    with reverts("Must keep fees at 10% or less"):
        artify.updateBuyFees(5, 5, 1, 1, {"from": owner})

    assert artify.buyOperationsFee() == 5
    assert artify.buyLiquidityFee() == 5
    assert artify.buyDevFee() == 0
    assert artify.buyBurnFee() == 0
    assert artify.buyTotalFees() == 10
    artify.updateBuyFees(1,2,3,4, {"from": owner})
    assert artify.buyOperationsFee() == 1
    assert artify.buyLiquidityFee() == 2
    assert artify.buyDevFee() == 3
    assert artify.buyBurnFee() ==  4
    assert artify.buyTotalFees() == 10

def test_update_sell_fees(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.updateSellFees(1, 2, 2, 4, {"from": other})
    with reverts("Must keep fees at 10% or less"):
        artify.updateSellFees(5, 5, 1, 1, {"from": owner})

    assert artify.sellOperationsFee() == 20
```

```python
    assert artify.sellLiquidityFee() == 20
    assert artify.sellDevFee() == 0
    assert artify.sellBurnFee() == 0
    assert artify.sellTotalFees() == 40
    artify.updateSellFees(1,2,3,4, {"from": owner})
    assert artify.sellOperationsFee() == 1
    assert artify.sellLiquidityFee() == 2
    assert artify.sellDevFee() == 3
    assert artify.sellBurnFee() == 4
    assert artify.sellTotalFees() == 10

def test_set_operations_address(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.setOperationsAddress(other, {"from": other})
    with reverts("_operationsAddress address cannot be 0"):
        artify.setOperationsAddress(ZERO_ADDRESS, {"from": owner})
    artify.setOperationsAddress(other, {"from": owner})

def test_set_dev_address(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.setDevAddress(other, {"from": other})
    with reverts("_devAddress address cannot be 0"):
        artify.setDevAddress(ZERO_ADDRESS, {"from": owner})
    artify.setDevAddress(other, {"from": owner})

def test_transfer_foreign_token(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)
    some_erc20 = deploy_test_erc20(owner)
    some_erc20.mint(owner, 1000, {"from": owner})

    #assert
    with reverts():
        artify.transferForeignToken(some_erc20.address, other, {"from":
other})
    with reverts("_token address cannot be 0"):
        artify.transferForeignToken(ZERO_ADDRESS, other, {"from": owner})
    with reverts("Can't withdraw native tokens"):
        artify.transferForeignToken(artify.address, other, {"from": owner})
    some_erc20.transfer(artify.address, 500, {"from": owner})
    artify.transferForeignToken(some_erc20.address, other, {"from": owner})

def test_treturn_to_normal_tax(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.returnToNormalTax({"from": other})

    assert artify.buyOperationsFee() == 5
    assert artify.buyLiquidityFee() == 5
    assert artify.buyDevFee() == 0
    assert artify.buyBurnFee() == 0
    assert artify.buyTotalFees() == 10
    assert artify.sellOperationsFee() == 20
    assert artify.sellLiquidityFee() == 20
    assert artify.sellDevFee() == 0
    assert artify.sellBurnFee() == 0
    assert artify.sellTotalFees() == 40
    artify.returnToNormalTax({"from": owner})
    assert artify.buyOperationsFee() == 4
    assert artify.buyLiquidityFee() == 1
    assert artify.buyDevFee() == 0
    assert artify.buyBurnFee() == 0
    assert artify.buyTotalFees() == 5
    assert artify.sellOperationsFee() == 3
    assert artify.sellLiquidityFee() == 2
    assert artify.sellDevFee() == 0
    assert artify.sellBurnFee() == 0
    assert artify.sellTotalFees() == 5

def test_exclude_from_max_transaction(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.excludeFromMaxTransaction(other, True, {"from": other})
    with reverts("Cannot remove uniswap pair from max txn"):
        artify.excludeFromMaxTransaction(artify.lpPair(), False, {"from":
owner})

    assert artify. isExcludedMaxTransactionAmount(other) == False
    artify.excludeFromMaxTransaction(other, True, {"from": owner})
    assert artify. isExcludedMaxTransactionAmount(other) == True

def test_enable_trading(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    #assert
    with reverts():
        artify.enableTrading(1, {"from": other})
    assert artify.tradingActive() == False
    artify.enableTrading(1, {"from": owner})
    assert artify.tradingActive() == True
    with reverts("Cannot reenable trading"):
        artify.enableTrading(1, {"from": owner})

def test_airdrop_to_wallets(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)
    #assert
    with reverts():
        artify.airdropToWallets([other], [10], {"from": other})
    with reverts("arrays must be the same length"):
        artify.airdropToWallets([other], [10, 20], {"from": owner})
    #with reverts("Can only airdrop 600 wallets per txn due to gas
limits"): // tested but to high array
        #artify.airdropToWallets([other]*10, [10]*10, {"from": owner})
    artify.airdropToWallets([other], [10], {"from": owner})

def test_set_automated_marker_pair(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    erc20 = deploy_test_erc20(owner)
    weth = deploy_test_weth(owner)
    factory = UniswapV2Factory.deploy(owner, {"from": owner})
    factory.createPair(erc20.address, weth.address)

    #assert
    with reverts():
        artify.setAutomatedMarketMakerPair(factory.address, True, {"from":
other})
    with reverts("The pair cannot be removed from
automatedMarketMakerPairs"):
        artify.setAutomatedMarketMakerPair(artify.lpPair(), True, {"from":
owner})

    assert artify.automatedMarketMakerPairs(factory.address) == False
    artify.setAutomatedMarketMakerPair(factory.address, True, {"from":
owner})
    assert artify.automatedMarketMakerPairs(factory.address) == True

def test_buy_back_tokens(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    artify = deploy_artify(owner)

    amount_wei = 1500
    #assert
    with reverts():
        artify.buyBackTokens(amount_wei, {"from": other})
    eleven_eth = 11000000000000000000
    with reverts("May not buy more than 10 ETH in a single buy to reduce
sandwich attacks"):
        artify.buyBackTokens(eleven_eth, {"from": owner})
```

6.0 Summary of the audit

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