Smart contract security audit

Bullz Challenge by WOM protocol

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During November of 2022, Wom Protocol 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. Wom Protocol team 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 WOM Protocol 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 Bullz Challenge contracts 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:

https://github.com/WOM-Protocol/Bullz-Challenge

- Commit ***6095eee4b40b5f27fe7008897ed783e95e2d5bca***

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.

3.4 Informational issues [1]

1. Architecture control

The bullz challenge platform allows users to create a contest using their own tokens/nfts. All calls to the main contract are made securely and information verifications are correct. Despite this, we recommend monitoring the platform by reviewing the different sales that occur to control unexpected behavior.

4.0 Testing phase - Coverage testing

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

```
contract: BullzMultipleExchange - 70.4%
  BullzMultipleExchange._addOffer - 92.9%
  BullzMultipleExchange._createBid - 79.2%
  BullzLibrary.computePlateformOwnerProfitByAmount - 75.0%
  BullzMultipleExchange.addOffer - 75.0%
  BullzMultipleExchange.buyOffer - 75.0%
  BullzMultipleExchange.delegateBuy - 75.0%
  FeeManager.setFeeTo - 75.0%
  TransferHelper.safeTransferETH - 75.0%
  TransferHelper.safeTransferFrom - 75.0%
  BullzMultipleExchange.acceptBid - 64.3%
  BullzMultipleExchange.cancelBid - 62.5%
  BullzMultipleExchange._buyOffer - 53.6%
  BullzMultipleExchange.setOfferPrice - 50.0%
  Ownable.transferOwnership - 0.0%

contract: BullzSingleExchange - 72.7%
  BullzSingleExchange._addOffer - 94.4%
  BullzSingleExchange.cancelOffer - 87.5%
  BullzSingleExchange._createBid - 77.8%
  BullzSingleExchange.addOffer - 75.0%
  BullzSingleExchange.buyOffer - 75.0%
  BullzSingleExchange.delegateBuy - 75.0%
  ERC721Validator._requireERC721 - 75.0%
  FeeManager.setFeeTo - 75.0%
  TransferHelper.safeTransferETH - 75.0%
  BullzSingleExchange.acceptBid - 63.1%
  BullzSingleExchange.cancelBid - 62.5%
  BullzSingleExchange._buyOffer - 52.4%
  BullzSingleExchange.setExpiresAt - 50.0%
  Ownable.transferOwnership - 0.0%
```

```
contract: ExchangeChallenge - 70.3%
  Challenge._getAirdropFeePercent - 100.0%
  Challenge.setFee - 100.0%
  Challenge.setMarketToken - 100.0%
  Challenge.setPrimaryToken - 100.0%
  ERC20Challenge.airdropTokenChallenge - 93.8%
  ERC20Challenge.addTokenChallenge - 91.7%
  Challenge.setPrimaryTokenPercent - 87.5%
  Challenge.setSecondaryTokenPercent - 87.5%
  Address.functionCallWithValue - 75.0%
  ERC20Challenge.withdrawTokenChallenge - 75.0%
  SafeERC20._callOptionalReturn - 75.0%
  Address.verifyCallResult - 62.5%
  Ownable.transferOwnership - 0.0%
```

5.0 Annexes

Bullz Multiple Exchange:

```
from brownie import reverts

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

from scripts.deploy import (
    deploy_test_erc20,
    deploy_test_erc1155,
    deploy_bullz_multiple_exchange
)

def test_add_offer(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    test_erc20.mint(account3, 300)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
    is_for_auction = False
```

```
    asset_id = 1
    price = 100

    with reverts():
        bse.addOffer([ZERO_ADDRESS, asset_id, test_erc20.address, price,
20, is_for_sell,
            is_for_auction, expires_at, 1, 100], {"from": account1})

    with reverts("Marketplace: Token address is not valid"):
        bse.addOffer([test_erc1155.address, asset_id, ZERO_ADDRESS, price,
20, is_for_sell,
            is_for_auction, expires_at, 1, 100], {"from": account1})

    with reverts("Marketplace: Price must be greater than zero"):
        bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
0, 20, is_for_sell,
            is_for_auction, expires_at, 1, 100], {"from": account1})

    with reverts("Marketplace: Amount must be greater than zero"):
        bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 0, is_for_sell,
            is_for_auction, expires_at, 1, 100], {"from": account1})

    with reverts("Marketplace: invalid expire time"):
        bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
            is_for_auction, 0, 1, 100], {"from": account1})

    with reverts("Contract not approved"):
        bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
            is_for_auction, expires_at, 1, 100], {"from": account1})

    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    assert tx.events['Listed'] is not None
    offerId = tx.events['Listed']['offerId']
    assert bse.offers(offerId)['seller'] == account1

def test_set_offer_price(only_local):
```

```
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})

    expires_at = get_timestamp()

    is_for_sell = True
    is_for_auction = False
    asset_id = 1
    price = 150
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    offerId = tx.events['Listed']['offerId']
    eventIdSetOfferPrice = 201
    bse.setOfferPrice(offerId, 200, eventIdSetOfferPrice, {"from":
account1})
    assert bse.offers(offerId)['price'] == 200

def test_set_expires_at(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})

    expires_at = get_timestamp()
```

```
    is_for_sell = True
    is_for_auction = False
    asset_id = 1
    price = 150
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    offerId = tx.events['Listed']['offerId']
    new_expires_at = get_timestamp(2)
    eventIdSetExpireAt = 202
    with reverts():
        bse.setExpiresAt(offerId, new_expires_at, eventIdSetExpireAt,
{"from": account2})
    bse.setExpiresAt(offerId, new_expires_at, eventIdSetExpireAt, {"from":
account1})
    assert bse.offers(offerId)['expiresAt'] == new_expires_at

def test_create_bid(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})

    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 150
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    offerId = tx.events['Listed']['offerId']
    eventIdBidCreated = 203
```

```
    with reverts("NFT Marketplace: Allowance error"):
        bse.safePlaceBid(offerId, price, 20, eventIdBidCreated, {"from":
account2})

    test_erc20.approve(bse.address, price, {"from": account2})
    bse.safePlaceBid(offerId, price, 20, eventIdBidCreated, {"from":
account2})
    assert bse.bidforAuctions(offerId, account2)['price'] == price

def test_cancel_bid(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})

    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 150
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    offerId = tx.events['Listed']['offerId']
    eventIdBidCreated = 203
    test_erc20.approve(bse.address, price, {"from": account2})
    bse.safePlaceBid(offerId, price, 20, eventIdBidCreated, {"from":
account2})
    eventIdBidCancelled = 204
    bse.cancelBid(offerId, account2, eventIdBidCancelled, {"from":
account2})
    assert bse.bidforAuctions(offerId, account2)['price'] == 0

def test_accept_bid(only_local):
```

```
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    test_erc20.mint(account3, 100)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})

    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 100
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    offerId = tx.events['Listed']['offerId']
    test_erc20.approve(bse.address, price, {"from": account2})
    test_erc20.approve(bse.address, price, {"from": account3})
    eventIdBidCreated = 203
    bse.safePlaceBid(offerId, price, 10, eventIdBidCreated, {"from":
account2})
    bse.safePlaceBid(offerId, price, 10, eventIdBidCreated, {"from":
account3})
    eventIdBidSuccessful = 207
    bse.acceptBid(offerId, account2, eventIdBidSuccessful, {"from":
account1})
    assert test_erc20.balanceOf(account1) == (price * 2 - (price * 1) /
100)
    assert test_erc20.balanceOf(account2) == 0
    assert test_erc20.balanceOf(account3) == price

def test_accept_multiple_bid(only_local):
    # arrange
```

```
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    test_erc20.mint(account3, 100)
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})

    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 100
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account1})
    offerId = tx.events['Listed']['offerId']
    test_erc20.approve(bse.address, price, {"from": account2})
    test_erc20.approve(bse.address, price, {"from": account3})

    eventIdBidCreated = 210
    bse.safePlaceBid(offerId, price, 10, eventIdBidCreated, {"from":
account2})
    eventIdBidSuccessful = 211
    bse.acceptBid(offerId, account2, eventIdBidSuccessful, {"from":
account1})
    eventIdBidCreated = 212
    bse.safePlaceBid(offerId, price, 10, eventIdBidCreated, {"from":
account3})
    eventIdBidSuccessful = 213
    bse.acceptBid(offerId, account3, eventIdBidSuccessful, {"from":
account1})

    assert test_erc20.balanceOf(account1) == price * 3 - (price * 2) / 100
    assert test_erc20.balanceOf(account2) == 0
```

```
    assert test_erc20.balanceOf(account3) == 0


def test_buy_direct_offer(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
    is_for_auction = False
    asset_id = 2
    price = 5 * ONE_ETH
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account2})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account2})
    offerId = tx.events['Listed']['offerId']
    event_id_swapped = 214
    bse.buyOffer(offerId, 1, event_id_swapped, { "from": account3, "value":
6 * ONE_ETH })
    assert test_erc1155.balanceOf(account3, asset_id) == 1

def test_delegate_buy_offer(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    account4 = get_account(4)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
```

```
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
    is_for_auction = False
    asset_id = 2
    price = 5 * ONE_ETH
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account2})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
        is_for_auction, expires_at, 1, 100], {"from": account2})
    offerId = tx.events['Listed']['offerId']
    event_id_swapped = 215
    bse.delegateBuy(offerId, 1, account4, event_id_swapped, { "from":
account3, "value": 6 * ONE_ETH })
    assert test_erc1155.balanceOf(account4, asset_id) == 1

def test_cancel_offer(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
    bse = deploy_bullz_multiple_exchange(owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
    is_for_auction = False
    asset_id = 2
    price = 5 * ONE_ETH
    test_erc1155.setApprovalForAll(bse.address, True, {"from": account2})
    tx = bse.addOffer([test_erc1155.address, asset_id, test_erc20.address,
price, 20, is_for_sell,
```

```
        is_for_auction, expires_at, 1, 100], {"from": account2})
    offerId = tx.events['Listed']['offerId']
    event_cancel_offer = 208
    with reverts("Offer should be expired"):
        bse.cancelOffer(offerId, event_cancel_offer, {"from": account2})
    evm_increase_time(172800) # increase blockchain timestamp 2 days
    bse.cancelOffer(offerId, event_cancel_offer, {"from": account2})

def test_set_owner_share(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    # deploy
    bse = deploy_bullz_multiple_exchange(owner)

    with reverts():
        bse.setFeeTo(1, 1, {"from": account1})
    bse.setFeeTo(1, 1, {"from": owner})
    assert bse.shares(1) == 1
```

Bullz Single Exchange:

```
from brownie import reverts

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

from scripts.deploy import (
    deploy_test_erc20,
    deploy_test_erc721,
    deploy_bullz_single_exchange
)

def test_add_offer(only_local):
    # arrange
    owner = get_account(0)
```

```
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    test_erc20.mint(account3, 300)
    test_erc721.awardItem(1, {"from": account1})
    test_erc721.awardItem(2, {"from": account2})

    expires_at = get_timestamp()

    is_for_sell = True
    is_for_auction = False
    with reverts("Marketplace: Seller address is not valid"):
        bse.addOffer(ZERO_ADDRESS, test_erc721.address, 1,
            test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})

    with reverts():
        bse.addOffer(account1, ZERO_ADDRESS, 1,
            test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})

    with reverts("Marketplace: Token address is not valid"):
        bse.addOffer(account2, test_erc721.address, 1,
            ZERO_ADDRESS, ONE_ETH, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account2})

    with reverts("Marketplace: Price must be greater than zero"):
        bse.addOffer(account2, test_erc721.address, 1,
            test_erc20.address, 0, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account2})

    with reverts("Marketplace: invalid expire time"):
        bse.addOffer(account2, test_erc721.address, 1,
            test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
            1, 1, {"from": account2})
```

```
    with reverts("Transfer caller is not owner"):
        bse.addOffer(account2, test_erc721.address, 1,
            test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account2})

    with reverts("Contract not approved"):
        bse.addOffer(account1, test_erc721.address, 1,
            test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})

    test_erc721.setApprovalForAll(bse.address, True, {"from": account1})
    tx = bse.addOffer(account1, test_erc721.address, 1,
            test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})
    assert tx.events['Listed'] is not None
    assert bse.offers(test_erc721.address, 1)['seller'] == account1

    with reverts("Offer exists already"):
        bse.addOffer(account1, test_erc721.address, 1,
                test_erc20.address, ONE_ETH, is_for_sell, is_for_auction,
                expires_at, 1, {"from": account1})

def test_set_offer_price(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc721.awardItem(1, {"from": account1})

    expires_at = get_timestamp()

    is_for_sell = True
    is_for_auction = False
    asset_id = 1
    price = 150
    test_erc721.setApprovalForAll(bse.address, True, {"from": account1})
```

```
    bse.addOffer(account1, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})
    bse.setOfferPrice(test_erc721.address, asset_id, 300, {"from":
account1})
    assert bse.offers(test_erc721.address, 1)['price'] == 300

def test_set_expires_at(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc721.awardItem(1, {"from": account1})

    expires_at = get_timestamp()

    is_for_sell = True
    is_for_auction = False
    asset_id = 1
    price = 150
    test_erc721.setApprovalForAll(bse.address, True, {"from": account1})
    bse.addOffer(account1, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})
    new_expires_at = get_timestamp(2)
    with reverts():
        bse.setExpiresAt(test_erc721.address, asset_id, new_expires_at,
{"from": account2})
    bse.setExpiresAt(test_erc721.address, asset_id, new_expires_at,
{"from": account1})
    assert bse.offers(test_erc721.address, 1)['expiresAt'] ==
new_expires_at

def test_create_bid(only_local):
    # arrange
    owner = get_account(0)
```

```
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc721.awardItem(1, {"from": account1})

    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 150
    test_erc721.setApprovalForAll(bse.address, True, {"from": account1})
    bse.addOffer(account1, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})
    with reverts("NFT Marketplace: Allowance error"):
        bse.safePlaceBid(test_erc721.address, asset_id, test_erc20.address,
150, {"from": account2})

    test_erc20.approve(bse.address, price, {"from": account2})
    bse.safePlaceBid(test_erc721.address, asset_id, test_erc20.address,
150, {"from": account2})
    assert bse.bidforAuctions(test_erc721.address, asset_id,
account2)['price'] == price

def test_cancel_bid(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc721.awardItem(1, {"from": account1})
```

```
    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 150
    test_erc721.setApprovalForAll(bse.address, True, {"from": account1})
    bse.addOffer(account1, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})
    test_erc20.approve(bse.address, price, {"from": account2})
    bse.safePlaceBid(test_erc721.address, asset_id, test_erc20.address,
150, {"from": account2})
    bse.cancelBid(test_erc721.address, asset_id, account2, {"from":
account2})
    assert bse.bidforAuctions(test_erc721.address, asset_id,
account2)['price'] == 0

def test_accept_bid(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    test_erc20.mint(account3, 100)
    test_erc721.awardItem(1, {"from": account1})

    expires_at = get_timestamp()
    is_for_sell = False
    is_for_auction = True
    asset_id = 1
    price = 100
    test_erc721.setApprovalForAll(bse.address, True, {"from": account1})
    bse.addOffer(account1, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account1})
```

```
    test_erc20.approve(bse.address, price, {"from": account2})
    test_erc20.approve(bse.address, price, {"from": account3})
    bse.safePlaceBid(test_erc721.address, asset_id, test_erc20.address,
100, {"from": account2})
    bse.safePlaceBid(test_erc721.address, asset_id, test_erc20.address,
100, {"from": account3})
    bse.acceptBid(test_erc721.address, asset_id, account2, {"from":
account1})
    assert test_erc20.balanceOf(account1) == (price * 2 - (price * 1) /
100)
    assert test_erc20.balanceOf(account2) == 0
    assert test_erc20.balanceOf(account3) == price

def test_buy_direct_offer(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc721.awardItem(1, {"from": account1})
    test_erc721.awardItem(2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
    is_for_auction = False
    asset_id = 2
    price = 5 * ONE_ETH
    test_erc721.setApprovalForAll(bse.address, True, {"from": account2})
    bse.addOffer(account2, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account2})
    bse.buyOffer(test_erc721.address, asset_id, { "from": account3,
"value": 6 * ONE_ETH })
    assert test_erc721.ownerOf(asset_id) == account3

def test_delegate_buy_offer(only_local):
    # arrange
```

```
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    account4 = get_account(4)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc721.awardItem(1, {"from": account1})
    test_erc721.awardItem(2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
    is_for_auction = False
    asset_id = 2
    price = 5 * ONE_ETH
    test_erc721.setApprovalForAll(bse.address, True, {"from": account2})
    bse.addOffer(account2, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account2})
    bse.delegateBuy(test_erc721.address, asset_id, account4, { "from":
account3, "value": 6 * ONE_ETH })
    assert test_erc721.ownerOf(asset_id) == account4

def test_cancel_offer(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    account3 = get_account(3)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc721 = deploy_test_erc721(owner)
    bse = deploy_bullz_single_exchange(owner)

    test_erc721.awardItem(1, {"from": account1})
    test_erc721.awardItem(2, {"from": account2})

    expires_at = get_timestamp()
    is_for_sell = True
```

```
    is_for_auction = False
    asset_id = 2
    price = 5 * ONE_ETH
    test_erc721.setApprovalForAll(bse.address, True, {"from": account2})
    bse.addOffer(account2, test_erc721.address, asset_id,
            test_erc20.address, price, is_for_sell, is_for_auction,
            expires_at, 1, {"from": account2})
    with reverts():
        bse.cancelOffer(test_erc721.address, asset_id, {"from": account3})
    with reverts("Offer should be expired"):
        bse.cancelOffer(test_erc721.address, asset_id, {"from": account2})
    evm_increase_time(172800) # increase blockchain timestamp 2 days
    bse.cancelOffer(test_erc721.address, asset_id, {"from": account2})

def test_set_owner_share(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    # deploy
    bse = deploy_bullz_single_exchange(owner)

    with reverts():
        bse.setFeeTo(1, 1, {"from": account1})
    bse.setFeeTo(1, 1, {"from": owner})
    assert bse.shares(1) == 1
```

Test Challenge:

```
from brownie import reverts

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

from scripts.deploy import (
    deploy_test_erc20,
    deploy_test_erc1155,
    deploy_exchange_challenge
```

```
)

def test_set_fee_primary_token(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    with reverts():
        c.setPrimaryTokenPercent(10, {"from": account1})
    with reverts("Challenge Exchange: Percent must be between 0 to 100 with
2 decimal point value."):
        c.setPrimaryTokenPercent(0, {"from": owner})
    c.setPrimaryTokenPercent(10, {"from": owner})
    assert c.getAirdropFeePercent(c.primaryToken()) == 10

def test_set_fee_secondary_token(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    second_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})
```

```
    with reverts():
        c.setSecondaryTokenPercent(10, {"from": account1})
    with reverts("Challenge Exchange: Percent must be between 0 to 100 with
2 decimal point value."):
        c.setSecondaryTokenPercent(0, {"from": owner})
    c.setSecondaryTokenPercent(10, {"from": owner})
    assert c.getAirdropFeePercent(second_erc20.address) == 10

def test_set_primary_token(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    second_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    with reverts():
        c.setPrimaryToken(second_erc20.address, {"from": account1})
    with reverts("Challenge Exchange: Not a valid address"):
        c.setPrimaryToken(ZERO_ADDRESS, {"from": owner})
    c.setPrimaryToken(second_erc20.address, {"from": owner})
    assert c.primaryToken() == second_erc20.address

def test_set_fee(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    second_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)
```

```
    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    with reverts():
        c.setFee(1, {"from": account1})
    with reverts("Challenge Exchange: Fee must be greated than zero."):
        c.setFee(0, {"from": owner})
    c.setFee(50, {"from": owner})
    assert c.bullzFee() == 50

def test_market_token(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    second_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    with reverts():
        c.setMarketToken(second_erc20.address, {"from": account1})
    with reverts("Challenge Exchange: Not a valid address"):
        c.setMarketToken(ZERO_ADDRESS, {"from": owner})
    c.setMarketToken(second_erc20.address, {"from": owner})
    assert c.marketToken() == second_erc20.address

def test_create_challenge(only_local):
    # arrange
    owner = get_account(0)
```

```
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)

    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    event_id_add_token_challenge = 4
    event_id_air_drop_token_challenge = 5
    event_id_withdraw_token_challenge = 6

    winner_count = 1
    token_amount = 2
    airdrop_start = get_timestamp()
    airdrop_end  = get_timestamp(3)
    with reverts("Challenge Exchange: Token address not valid"):
        c.addTokenChallenge(ZERO_ADDRESS, winner_count, token_amount,
airdrop_start,
            airdrop_end, event_id_add_token_challenge, {"from": owner})
    with reverts("Winner count must be upper to 0"):
        c.addTokenChallenge(test_erc20.address, 0, token_amount,
airdrop_start,
            airdrop_end, event_id_add_token_challenge, {"from": owner})

    with reverts():
        c.addTokenChallenge(test_erc20.address, winner_count, token_amount,
airdrop_start,
            airdrop_end, event_id_add_token_challenge, {"from": account1})

    airdrop_fee = winner_count * token_amount *
(c.getAirdropFeePercent(test_erc20.address) / 100)
    total_payable = winner_count * token_amount * airdrop_fee
    test_erc20.approve(c.address, total_payable * 1e18, {"from": account1})

    with reverts("Challenge Exchange: invalid start at airdrop"):
        c.addTokenChallenge(test_erc20.address, winner_count, token_amount,
```

```
0,
            airdrop_end, event_id_add_token_challenge, {"from": account1})
    with reverts("Challenge Exchange: invalid end at airdrop"):
        c.addTokenChallenge(test_erc20.address, winner_count, token_amount,
airdrop_start,
            0, event_id_add_token_challenge, {"from": account1})

    tx = c.addTokenChallenge(test_erc20.address, winner_count,
token_amount, airdrop_start,
            airdrop_end, event_id_add_token_challenge, {"from": account1})
    assert tx.events['AddTokenChallenge'] is not None
    challenge_id = tx.events['AddTokenChallenge']['challengeId']

    with reverts("Challenge Exchange: Receiver address not valid"):
        c.airdropTokenChallenge(challenge_id, ZERO_ADDRESS,
event_id_air_drop_token_challenge, {"from": account1})

    with reverts("Challenge Exchange: caller not an owner"):
        c.airdropTokenChallenge(challenge_id, account2,
event_id_air_drop_token_challenge, {"from": account2})

    with reverts("Challenge Exchange: invalid start at airdrop"):
        c.airdropTokenChallenge(challenge_id, account2,
event_id_air_drop_token_challenge, {"from": account1})

    evm_increase_time(172800) # increase 2 days
    c.airdropTokenChallenge(challenge_id, account2,
event_id_air_drop_token_challenge, {"from": account1})

def test_withdraw_token_challenge(only_local):
    # arrange
    owner = get_account(0)
    account1 = get_account(1)
    account2 = get_account(2)
    # deploy
    test_erc20 = deploy_test_erc20(owner)
    test_erc1155 = deploy_test_erc1155(owner)

    test_erc20.mint(account1, 100)
    test_erc20.mint(account2, 100)
    c = deploy_exchange_challenge(test_erc20.address, owner)
```

```
    test_erc1155.awardItem(1, 300, "0x", 0, 2, {"from": account1})
    test_erc1155.awardItem(2, 300, "0x", 0, 2, {"from": account2})

    event_id_add_token_challenge = 4
    event_id_air_drop_token_challenge = 5
    event_id_withdraw_token_challenge = 6

    winner_count = 2
    token_amount = 2
    airdrop_start = get_timestamp()
    airdrop_end  = get_timestamp(3)

    airdrop_fee = token_amount *
(c.getAirdropFeePercent(test_erc20.address) / 10000)
    total_payable = winner_count * token_amount * airdrop_fee
    test_erc20.approve(c.address, total_payable * 1e18, {"from": account1})

    tx = c.addTokenChallenge(test_erc20.address, winner_count,
token_amount, airdrop_start,
            airdrop_end, event_id_add_token_challenge, {"from": account1})
    assert tx.events['AddTokenChallenge'] is not None
    challenge_id = tx.events['AddTokenChallenge']['challengeId']

    evm_increase_time(172800) # increase 2 days
    with reverts("Challenge exchange: airdrop not ended"):
        c.withdrawTokenChallenge(challenge_id,
event_id_withdraw_token_challenge, {"from": account1})

    evm_increase_time(172800) # increase 2 days
    c.withdrawTokenChallenge(challenge_id,
event_id_withdraw_token_challenge, {"from": account1})
```

6.0 Summary of the audit

The contract follows all the recommendations for secure programming & all the good coding practice guidelines recommended by solidity. The control structures established in the different functions are correct and it does not leave ambiguity in the functions. Calls to external contracts are controlled and are made securely.

As a summary, the contract has been developed in a safe way and from CTDSEC we recommend monitoring the platform and establishing a reporting system for users in case of finding any error in the platform.