Smart contract security audit

Wojak Cash

v.1.1

Table of Contents

1.0 Introduction

1.1 Project engagement

During May of 2023, Wojak Cash 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. Wojak Cash 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 Wojak Cash 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 Wojak Cash 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:

Bogdanoff.sol [SHA256] - 5b16ff0624335cf139382af56fa1860ee67a772fe19ce188261554fb84b886b3

Fixed version:

Bogdanoff_final.sol [SHA256] -

89a816200d57919c4a13f72236d5d1b4b7d217612cfd8b54353a432f81864d1d

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 [2]

1. Balance Deduction Error in transfer() Function Leading to Incorrect Fee Calculation

Function: Transfer()

Issue: The deducted transfer amount from the sender's balance is incorrect because it is subtracted after deducting fees. As a result, the sender ends up with the fee amount after the transfer.

Fix: To address this issue, update the code by adding the following line: _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); before the fee deduction, where amount is subtracted by fee. This ensures that the sender's balance is correctly updated to reflect the transferred amount minus the fee. Alternatively, you can modify the transfer logic to handle this issue appropriately.

2. Reverted Transactions and Redundant Checks in transfer() Function

Function: Transfer()

Issue: The transfer operation fails when _BogdanoffHasPhone is true and shouldCallBogdanoff() is not true in the callBogdanoff() function. This frequently results in the transfer between two addresses being reverted due to a function. Moreover, the shouldCallBogdanoff() function is called twice within the _transfer function.

Fix: One possible solution is to eliminate the use of the shouldCallBodanoff() require statement inside the _transfer function. Alternatively, you can modify the logic of the _transfer function. If the _transfer function is only called by itself, you can consider making it an internal function instead.

3. Balances can be stuck forever

Function: *

Issue: Bogdanoff is accumulating tokens/bnb and buying/selling them depending on the probability that is executed in each transaction. There is a chance that Bogdanoff accumulate tokens/bnbs that will never use all of them. We recommend checking/changing the logic to rebalance the funds that Bogdanoff is holding.

Fix: Add a function that rebalance the tokens/bnbs that bogdanoff is holding.

3.2 Medium severity issues [1]

1. Logic error - Uniswap Error INVALID_TO Despite Sufficient Funds in swapEthForTokens() Function

Function: swapEthForTokens()

Problem: When the function callBogdanoff() is executed and a purchase is made, Uniswap returns the error "UniswapV2: INVALID_TO" despite the contract having sufficient funds.

Solution: Either assign a different wallet account to the third argument parameter in the swapExactETHForTokensSupportingFeeOnTransferTokens function, or create an intermediate contract that handles the swapping of ETH for tokens. For instance, consider using owner() instead of address(this).

3.3 Low severity issues [0]

No low severity issues found.

3.4 Informational issues [1]

1. Possible transaction reverse

Function: _transfer()

Problem: When the callBogdanoff() function is triggered and there are insufficient funds in the contract to perform a buy/sell transaction, the transaction between addresses will be reverted.

Solution: Make sure that the contract always maintains sufficient funds.

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.*

Wojak Cash tests

```
tests/test_demo.py::test_transfer RUNNING
Transaction sent: 0xf1c0e6ba69f3ab8dec2223ffa0f5601323d1d50b6d6356cfa6a0a89bbe699a80
Transaction sent: 0x0f96cf5f9ba50114ada7127e4da7a792d5eb71b94276bec7462066bc171d4cab
Transaction sent: 0xc3a846c436065d8f55ee09321b855f2b70197eb708304a8e5f95f8206ba77408
tests/test_demo.py::test_transfer PASSED
tests/test_demo.py::test_transfer_with_swap_and_liquify PASSED
tests/test_demo.py::test_call_origin RUNNING
Transaction sent: 0x53d0c152d71e500a805ddfb6eb75c7862013801fc1749d7e2edddd4a44bb1881
Transaction sent: 0xf4e2396eb5ca2c294885377b5b657f75dc71d35a222fce559b39bc73c00d8a5d
tests/test_demo.py::test_call_origin PASSED
tests/test_demo.py::test_set_marketing_wallet RUNNING
Transaction sent: 0xa26c8173190124f9e903fd5c270e1ce059117eae9acfbc2cbef29b428a9b3960
Transaction sent: 0xcf0f5a8d9d7c703035e823beee362e8bffd27c8a1832d439f7adcaf3fa15b35c
tests/test_demo.py::test_set_marketing_wallet PASSED
tests/test_demo.py::test_set_fee RUNNING
Transaction sent: 0xd76bd44c15f21580d2b9ed9f493eae7449c3159a8abf64a0c695a0bfa9c26375
Transaction sent: 0xa71b28b28fae9ba417d5bc322e0511e93ce7f53ecb6bc49cd1fe3810c204656c
tests/test_demo.py::test_set_fee PASSED
tests/test_demo.py::test_set_call_wait_time RUNNING
Transaction sent: 0x8f4b7b70b31e2cf9e74d6913ec3ea7edabc531d7299fcbf58129a430384e55d8
Transaction sent: 0x12609df52f4db2f1629a54dfb6c9343e17cba3459fd9db39405011500c2ca9f8
tests/test_demo.py::test_set_call_wait_time PASSED

========================================================================== 6 passed in 17.01s
```

5.0 Annexes

Wojak Cash testing:

```
from brownie import (
    reverts
)

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

from scripts.deploy import (
    deploy_demo
)

def test_transfer(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    external_account = get_account(2)
    demo = deploy_demo(owner)

    #assert
    with reverts("ERC20: transfer from the zero address"):
        demo.transfer(other_account, 1000, {"from": ZERO_ADDRESS})

    with reverts("ERC20: transfer to the zero address"):
        demo.transfer(ZERO_ADDRESS, 1000, {"from": owner})

    with reverts("ERC20: transfer amount exceeds balance"):
        demo.transfer(owner, 1000, {"from": other_account})

    # test normal transfer with whitelisted sender
    tx = demo.transfer(other_account, 1000, {"from": owner})
    assert tx.events["Transfer"] is not None
```

```
    assert tx.events["Transfer"]["from"] == owner
    assert tx.events["Transfer"]["to"] == other_account
    assert tx.events["Transfer"]["value"] == 1000
    assert demo.balanceOf(other_account) == 1000

    # test normal transfer with whitelisted to
    tx = demo.transfer(owner, 500, {"from": other_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"]["from"] == other_account
    assert tx.events["Transfer"]["to"] == owner
    assert tx.events["Transfer"]["value"] == 500
    assert demo.balanceOf(other_account) == 500

    # test normal transfer without whitelisted accounts and with fees 5%
    tx = demo.transfer(external_account, 500, {"from": other_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == other_account
    assert tx.events["Transfer"][0]["to"] == demo.address
    assert tx.events["Transfer"][0]["value"] == 500 * 0.05
    assert tx.events["Transfer"][1]["from"] == other_account
    assert tx.events["Transfer"][1]["to"] == external_account
    assert tx.events["Transfer"][1]["value"] == 500 - (500 * 0.05)
    # assert demo.balanceOf(other_account) == 0 # FIX: fix still pending
    assert demo.balanceOf(external_account) == 500 - (500 * 0.05)

def test_transfer_with_swap_and_liquify(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    external_account = get_account(2)
    demo = deploy_demo(owner)

    # send some tokens to another account not excluded from fees
    demo.transfer(other_account, 20000, {"from": owner})
    # decrase numTokensToSell to force swapAndLiquify
    demo.setNumTokensSellToAddToLiquidity(200)

    # assert
    tx = demo.transfer(external_account, 5000, {"from": other_account})
    assert tx.events["Transfer"][1]["value"] == 5000 - (5000 * 0.05)

    tx = demo.transfer(external_account, 1000, {"from": other_account})
```

```
    assert tx.events["Transfer"] is not None

def test_call_Bogdanoff(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    other_account = get_account(2)
    external_account = get_account(3)
    demo = deploy_demo(owner)

    #owner.transfer(to=demo.address, amount=1e18)

    # make some transactions to have contract balance with fees
    demo.transfer(other_account, 20000, {"from": owner})
    demo.transfer(external_account, 5000, {"from": other_account})

    # assert
    with reverts():
        demo.giveDemoBogdanoffAPhone({"from": not_owner})

    # if ! DemoBogdanoffHasPhone
    r = demo.shouldCallDemoBogdanoff({"from": owner})
    assert r == False

    assert demo._DemoBogdanoffHasPhone() == False
    assert demo._lastCall() == 0
    demo.giveDemoBogdanoffAPhone({"from": owner})
    assert demo._DemoBogdanoffHasPhone() == True
    assert demo._lastCall() != 0

    with reverts("DemoBogdanoff already has a phone"):
        demo.giveDemoBogdanoffAPhone({"from": owner})

    # if !block.timestamp < _lastCall + _waitTime)
    r = demo.shouldCallDemoBogdanoff({"from": owner})
    assert r == False

    evm_increase_time(86400) # increase timestamp 1 day

    for _ in range(25):
        owner.transfer(to=demo.address, amount=1e18) # transfer some funds
to contract
```

```
        tx = demo.transfer(external_account, 50, {"from": other_account})
        evm_increase_time(86400) # increase timestamp 1 day
        evm_increase_block(1)

def test_set_marketing_wallet(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    new_marketing = get_account(2)
    demo = deploy_demo(owner)

    # assert
    with reverts():
        demo.setMarketingWallet(new_marketing, {"from": not_owner})
    with reverts("Incorrect address."):
        demo.setMarketingWallet(ZERO_ADDRESS, {"from": owner})

    assert demo._marketingWallet() == owner
    demo.setMarketingWallet(new_marketing, {"from": owner})
    assert demo._marketingWallet() == new_marketing

def test_set_fee(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    demo = deploy_demo(owner)

    new_fee = 90
    # assert
    with reverts():
        demo.setFee(80, {"from": not_owner})
    with reverts("Fee must be lower than 10%"):
        demo.setFee(110, {"from": owner})

    assert demo._fee() == 50
    demo.setFee(new_fee, {"from": owner})
    assert demo._fee() == new_fee

def test_set_call_wait_time(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
```

```
    demo = deploy_demo(owner)

    # assert
    with reverts():
        demo.setCallWaitTime(80, {"from": not_owner})
    with reverts("Call wait time can't be more than 1 day"):
        demo.setCallWaitTime(90000, {"from": owner})

    assert demo._waitTime() == 900 # default
    demo.setCallWaitTime(950, {"from": owner})
    assert demo._waitTime() == 950
```

6.0 Summary of the audit

High/medium vulnerabilities were found, and the development team applied patches to all of them before deploying the contract.

Fixed vulnerabilities before the deployment at file version:

Bogdanoff_final.sol [SHA256] -

89a816200d57919c4a13f72236d5d1b4b7d217612cfd8b54353a432f81864d1d