Smart contract security audit Cbetchip

v.1.2

Table of Contents

1.0 Introduction

1.1 Project engagement

During April of 2023, Cbetchip 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. Cbetchip 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 Cbetchip 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 Cbetchip 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>

IExchange_latest.sol [SHA256] -

b2b075ced6a1395241a0cb8d0f892db576312fb837ef9bba0629cb30454046f6

Cbetchip.sol [SHA256] - 5f4701d0d2475341f9b31c47bac197480facf70557e2045016636fa87d5d910f

Fixed version:

https://arbiscan.io/address/0x5de2672d115f3f489cFeEAdAEeABD10119036EBA#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 [1 - Fixed]

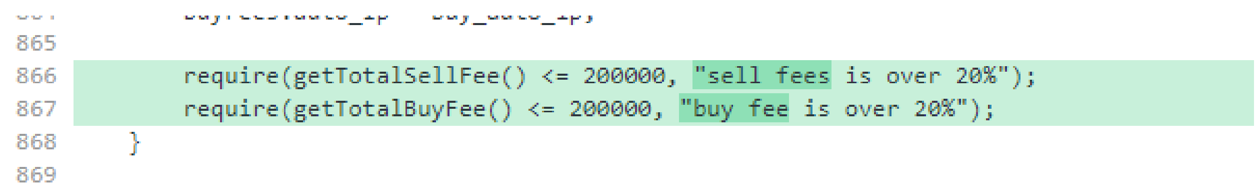
1. Require descriptions could cause confusion

Function: setFees()

Issue: The usage of identical require messages for multiple requirements could cause confusion in identifying the specific error that occurred.

Solution: Use distinct require messages such as "sell fees is over 20%" for require(getTotalSellFee() <= 200000) and "buy fee is over 20%" for require(getTotalBuylFee() <= 200000).

Dev fix:



```
        require(getTotalSellFee() <= 200000, "sell fees is over 20%");
        require(getTotalBuyFee() <= 200000, "buy fee is over 20%");
    }
```

3.4 Informational issues [1 - Fixed]

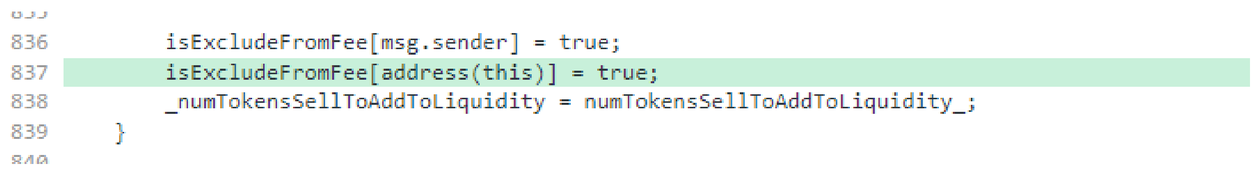
1. Constructor contract not excluded from fees

Function: constructor()

Issue: It is unclear whether the contract should be excluded from fees or not.

Solution: It is strongly recommended to set isExcludeFromFee[address(this)] = true to ensure the contract is excluded from fees.

Dev fix:

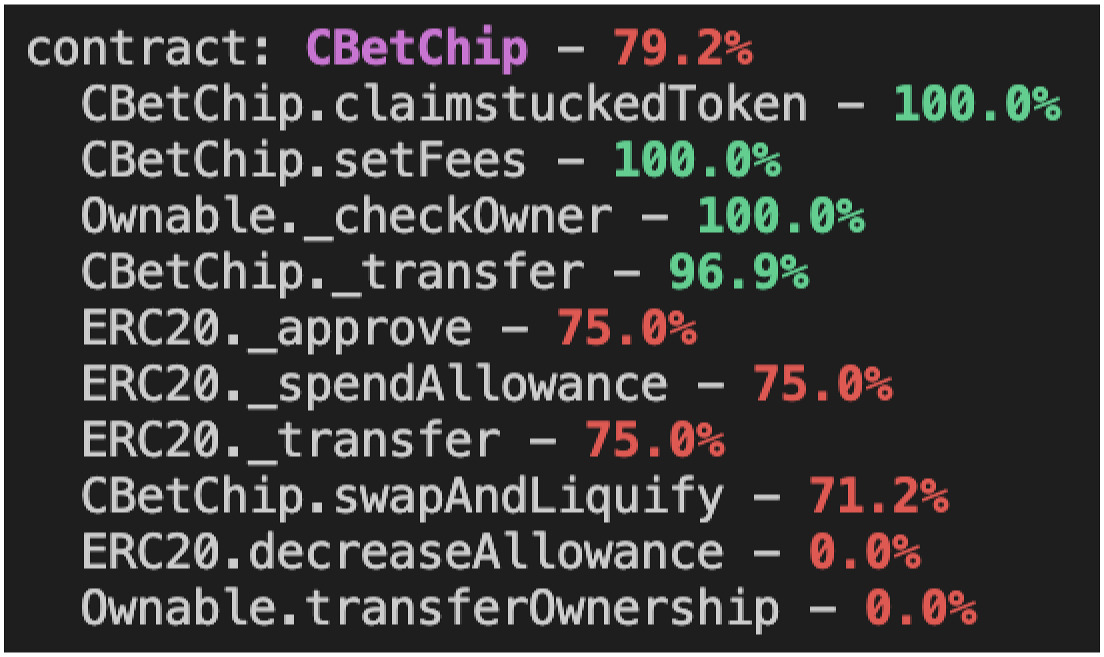


```
        isExcludeFromFee[msg.sender] = true;
        isExcludeFromFee[address(this)] = true;
        _numTokensSellToAddToLiquidity = numTokensSellToAddToLiquidity_;
    }
```

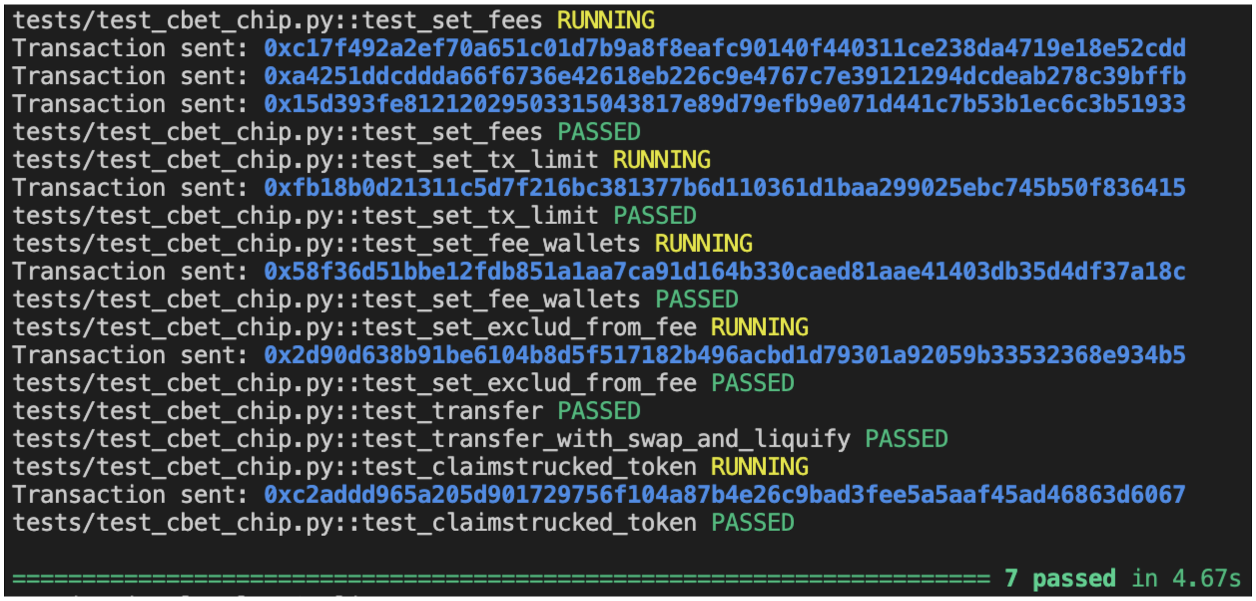
4.0 Testing coverage - python

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>CBETCHIP tests</u>



```
contract: CBetChip - 79.2%
  CBetChip.claimstuckedToken - 100.0%
  CBetChip.setFees - 100.0%
  Ownable._checkOwner - 100.0%
  CBetChip._transfer - 96.9%
  ERC20._approve - 75.0%
  ERC20._spendAllowance - 75.0%
  ERC20._transfer - 75.0%
  CBetChip.swapAndLiquify - 71.2%
  ERC20.decreaseAllowance - 0.0%
  Ownable.transferOwnership - 0.0%
```



```
tests/test_cbet_chip.py::test_set_fees RUNNING
Transaction sent: 0xc17f492a2ef70a651c01d7b9a8f8eafc90140f440311ce238da4719e18e52cdd
Transaction sent: 0xa4251ddcddda66f6736e42618eb226c9e4767c7e39121294dcdeab278c39bffb
Transaction sent: 0x15d393fe812120295033150438 17e89d79efb9e071d441c7b53b1ec6c3b51933
tests/test_cbet_chip.py::test_set_fees PASSED
tests/test_cbet_chip.py::test_set_tx_limit RUNNING
Transaction sent: 0xfb18b0d21311c5d7f216bc381377b6d110361d1baa299025ebc745b50f836415
tests/test_cbet_chip.py::test_set_tx_limit PASSED
tests/test_cbet_chip.py::test_set_fee_wallets RUNNING
Transaction sent: 0x58f36d51bbe12fdb851a1aa7ca91d164b330caed81aae41403db35d4df37a18c
tests/test_cbet_chip.py::test_set_fee_wallets PASSED
tests/test_cbet_chip.py::test_set_exclud_from_fee RUNNING
Transaction sent: 0x2d90d638b91be6104b8d5f517182b496acbd1d79301a92059b33532368e934b5
tests/test_cbet_chip.py::test_set_exclud_from_fee PASSED
tests/test_cbet_chip.py::test_transfer PASSED
tests/test_cbet_chip.py::test_transfer_with_swap_and_liquify PASSED
tests/test_cbet_chip.py::test_claimstrucked_token RUNNING
Transaction sent: 0xc2addd965a205d901729756f104a87b4e26c9bad3fee5a5aaf45ad46863d6067
tests/test_cbet_chip.py::test_claimstrucked_token PASSED

====================================================================== 7 passed in 4.67s
```

5.0 Annexes

CBETchip testing:

```
from brownie import (
    reverts
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
)

from scripts.deploy import (
    deploy_cbet_chip
)

def test_set_fees(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    cbet = deploy_cbet_chip(owner)

    # sell fees
    sell_markt_fee = 1000
    sell_reserve_fee = 1000
    sell_auto_lp = 2000
    # buy fees
    buy_marketing_fee = 2000
    buy_reserve_fee = 2000
    buy_auto_lp = 1000
    # assert
    with reverts():
        cbet.setFees(sell_markt_fee, sell_reserve_fee, sell_auto_lp,
                      buy_marketing_fee, buy_reserve_fee, buy_auto_lp,
{"from": not_owner})
    with reverts("fee is over 20%"):
        cbet.setFees(200000, sell_reserve_fee, sell_auto_lp,
                      buy_marketing_fee, buy_reserve_fee, buy_auto_lp,
{"from": owner})
```

```
    with reverts("fee is over 20%"):
        cbet.setFees(sell_markt_fee, sell_reserve_fee, sell_auto_lp,
                     200000, buy_reserve_fee, buy_auto_lp, {"from":
owner})

    cbet.setFees(sell_markt_fee, sell_reserve_fee, sell_auto_lp,
                     buy_marketing_fee, buy_reserve_fee, buy_auto_lp,
{"from": owner})

def test_set_tx_limit(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    cbet = deploy_cbet_chip(owner)

    num_tokens_sell_to_add_to_liquidity = 10000
    swap_and_liquify_enabled = False
    # assert
    with reverts():
        cbet.setTxLimit(num_tokens_sell_to_add_to_liquidity,
swap_and_liquify_enabled, {"from": not_owner})

    assert cbet._numTokensSellToAddToLiquidity() == 10000000e9 * 0.5 # set
in deploy
    assert cbet._swapAndLiquifyEnabled() == True # by default
    cbet.setTxLimit(num_tokens_sell_to_add_to_liquidity,
swap_and_liquify_enabled, {"from": owner})
    assert cbet._numTokensSellToAddToLiquidity() ==
num_tokens_sell_to_add_to_liquidity
    assert cbet._swapAndLiquifyEnabled() == swap_and_liquify_enabled

def test_set_fee_wallets(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    new_marketing = get_account(2)
    new_reserve = get_account(3)
    cbet = deploy_cbet_chip(owner)

    # assert
    with reverts():
        cbet.setFeeWallets(new_marketing, new_reserve, {"from": not_owner})
    assert cbet.feeWallets()[0] == ZERO_ADDRESS # marketing
    assert cbet.feeWallets()[1] == ZERO_ADDRESS # reserve
    cbet.setFeeWallets(new_marketing, new_reserve, {"from": owner})
    assert cbet.feeWallets()[0] == new_marketing
    assert cbet.feeWallets()[1] == new_reserve

def test_set_exclud_from_fee(only_local):
    #arrange
    owner = get_account(0)
    not_owner = get_account(1)
    new_excluded = get_account(2)
    cbet = deploy_cbet_chip(owner)

    # assert
    with reverts():
        cbet.setExcludFromFee(new_excluded, True, {"from": not_owner})

    assert cbet.isExcludeFromFee(new_excluded) == False
    cbet.setExcludFromFee(new_excluded, True, {"from": owner})
    assert cbet.isExcludeFromFee(new_excluded) == True

def test_transfer(only_local):
    #arrange
    owner = get_account(0)
    another_account = get_account(1)
    other_account = get_account(2)
    cbet = deploy_cbet_chip(owner)

    pair_addr = cbet.DexPair()
    num_tokens_sell_to_add_to_liquidity = 10000
    swap_and_liquify_enabled = False
    cbet.setTxLimit(num_tokens_sell_to_add_to_liquidity,
swap_and_liquify_enabled, {"from": owner})

    # assert

    # test normal transfer with excluded wallet
    tx1 = cbet.transfer(another_account, 1000, {"from": owner})
    assert tx1.events["Transfer"] is not None
    assert tx1.events["Transfer"]["from"] == owner
    assert tx1.events["Transfer"]["to"] == another_account
    assert tx1.events["Transfer"]["value"] == 1000

    # test sell transfer (without fees)
    tx2 = cbet.transfer(pair_addr, 500, {"from": another_account})
    assert tx2.events["Transfer"] is not None
    # fees transfer
    assert tx2.events["Transfer"][0]["from"] == another_account
    assert tx2.events["Transfer"][0]["to"] == cbet.address
    assert tx2.events["Transfer"][0]["value"] == 0
    # transfer
    assert tx2.events["Transfer"][1]["from"] == another_account
    assert tx2.events["Transfer"][1]["to"] == pair_addr
    assert tx2.events["Transfer"][1]["value"] == 500

    # test buy transfer (without fees)
    tx3 = cbet.transfer(another_account, 500, {"from": pair_addr})
    assert tx3.events["Transfer"] is not None
    # fees transfer
    assert tx3.events["Transfer"][0]["from"] == pair_addr
    assert tx3.events["Transfer"][0]["to"] == cbet.address
    assert tx3.events["Transfer"][0]["value"] == 0
    # transfer
    assert tx3.events["Transfer"][1]["from"] == pair_addr
    assert tx3.events["Transfer"][1]["to"] == another_account
    assert tx3.events["Transfer"][1]["value"] == 500

    # test normal transfer with excluded wallet (to)
    tx4 = cbet.transfer(owner, 500, {"from": another_account})
    assert tx4.events["Transfer"] is not None
    assert tx4.events["Transfer"]["from"] == another_account
    assert tx4.events["Transfer"]["to"] == owner
    assert tx4.events["Transfer"]["value"] == 500

    # sell fees
    sell_markt_fee = 1000
    sell_reserve_fee = 1000
    sell_auto_lp = 2000
    # buy fees
    buy_marketing_fee = 2000
    buy_reserve_fee = 2000
    buy_auto_lp = 1000
    cbet.setFees(sell_markt_fee, sell_reserve_fee, sell_auto_lp,
                      buy_marketing_fee, buy_reserve_fee, buy_auto_lp,
{"from": owner})

    # test sell transfer (with fees)
    tx5 = cbet.transfer(pair_addr, 500, {"from": another_account})
    assert tx5.events["Transfer"] is not None
    assert tx5.events["Transfer"][0]["from"] == another_account
    assert tx5.events["Transfer"][0]["to"] == cbet.address
    assert tx5.events["Transfer"][0]["value"] == 2
    assert tx5.events["Transfer"][1]["from"] == another_account
    assert tx5.events["Transfer"][1]["to"] == pair_addr
    assert tx5.events["Transfer"][1]["value"] == 498

    # test buy transfer (with fees)
    tx6 = cbet.transfer(another_account, 450, {"from": pair_addr})
    assert tx6.events["Transfer"] is not None
    assert tx6.events["Transfer"][0]["from"] == pair_addr
    assert tx6.events["Transfer"][0]["to"] == cbet.address
    assert tx6.events["Transfer"][0]["value"] == 2
    assert tx6.events["Transfer"][1]["from"] == pair_addr
    assert tx6.events["Transfer"][1]["to"] == another_account
    assert tx6.events["Transfer"][1]["value"] == 448

    # test normal transfer without excluded wallets
    tx7 = cbet.transfer(other_account, 400, {"from": another_account})
    assert tx7.events["Transfer"] is not None
    assert tx7.events["Transfer"]["from"] == another_account
    assert tx7.events["Transfer"]["to"] == other_account
    assert tx7.events["Transfer"]["value"] == 400

def test_transfer_with_swap_and_liquify(only_local):
    #arrange
    owner = get_account(0)
    another_account = get_account(1)
    new_marketing = get_account(3)
    new_reserve = get_account(4)
    cbet = deploy_cbet_chip(owner)

    pair_addr = cbet.DexPair()
    num_tokens_sell_to_add_to_liquidity = 1
    swap_and_liquify_enabled = True
    cbet.setTxLimit(num_tokens_sell_to_add_to_liquidity,
swap_and_liquify_enabled, {"from": owner})

    # test normal transfer with excluded wallet to have a bit of amount in 
another_account wallet
    cbet.transfer(another_account, 1000, {"from": owner})
    #cbet.transfer(cbet.address, 1000, {"from": owner})

    # sell fees
    sell_markt_fee = 4000
    sell_reserve_fee = 5000
    sell_auto_lp = 6000
    # buy fees
    buy_marketing_fee = 2000
    buy_reserve_fee = 2000
    buy_auto_lp = 1000
    cbet.setFees(sell_markt_fee, sell_reserve_fee, sell_auto_lp,
                  buy_marketing_fee, buy_reserve_fee, buy_auto_lp, 
{"from": owner})
    cbet.setFeeWallets(new_marketing, new_reserve, {"from": owner})

    # assert

    tx1 = cbet.transfer(pair_addr, 165, {"from": another_account})
    assert tx1.events["Transfer"] is not None
    assert tx1.events["Transfer"][0]["from"] == another_account
    assert tx1.events["Transfer"][0]["to"] == cbet.address
    assert tx1.events["Transfer"][0]["value"] == 2
    assert tx1.events["Transfer"][1]["from"] == another_account
    assert tx1.events["Transfer"][1]["to"] == pair_addr
    assert tx1.events["Transfer"][1]["value"] == 163

    # test normal transfer without excluded wallets and swap and liquify 
enabled
    tx2 = cbet.transfer(pair_addr, 300, {"from": another_account})
    assert tx2.events["Transfer"] is not None

def test_claimstrucked_token(only_local):
    #arrange
    owner = get_account(0)
    another_account = get_account(1)
    not_owner = get_account(2)
    cbet = deploy_cbet_chip(owner)
    cbet_2 = deploy_cbet_chip(owner)

    # assert
    with reverts():
        cbet.claimstuckedToken(cbet.address, 100, {"from": not_owner})

    # send some tokens to another contract token to recover
    cbet_2.transfer(cbet.address, 1000, {"from": owner})

    # Send one eth to the contract
    owner.transfer(to=cbet.address, amount=1e18)

    # Recover the eth
    cbet.claimstuckedToken(ZERO_ADDRESS, 100, {"from": owner})

    # Recover the tokens
    cbet.claimstuckedToken(cbet_2.address, 1000, {"from": owner})
```

6.0 Summary of the audit

No high/medium vulnerabilities were found, we recommend checking the informative/low ones before deploying.

Fixed vulnerabilities before the deployment [**scope of fixed version is the reported issues in v1*]. (https://arbiscan.io/address/0x5de2672d115f3f489cFeEAdAEeABD10119036EBA#code).