Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

OnlyPump

Date: 02/07/2025

Audit Status: PASS

Audit Edition: Advanced+

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

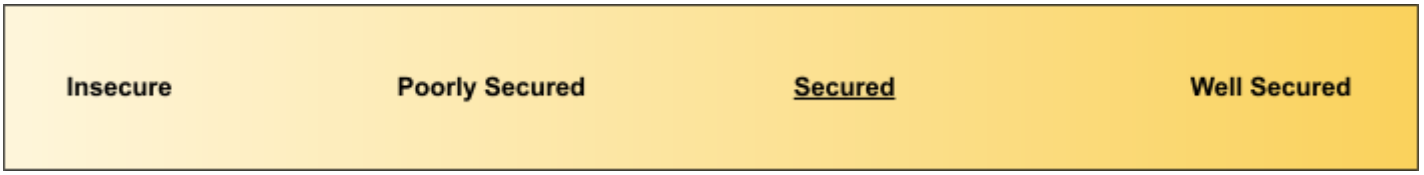
Risk Analysis

Vulnerability summary

Classification	Description
High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Secured.**



Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the OnlyPump contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	Token.sol [SHA256]: b81d91fc930072916b7c86f1f993cddacfa80c23c81a3078f262240f5d3b14f7 Master.sol [SHA256]: b4f2443c13bb69b12278af1d67c649fde890065c9eddeccfea796ec9c86bcdc5 — Fixed version - TokenFixed.sol [SHA256] b18ae6a2fa04824698bdab83103656c9f900b8983722186565b1f9b7a38c381a Fixed version v2 - TokenFinal.sol [SHA256] 4dfcd0ff0db50a8f27c3efbe701be3d585da9b105ddb425c2b587224882799b3
Audit Methodology	Static, Manual

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.

Category	Item
Code review & Functional Review	• Compiler warnings. • Race conditions and Reentrancy. Cross-function race conditions. • Possible delays in data delivery. • Oracle calls. • Front running. • Timestamp dependence. • Integer Overflow and Underflow. • DoS with Revert. • DoS with block gas limit. • Methods execution permissions. • Economy model. • Private user data leaks. • Malicious Event log. • Scoping and Declarations. • Uninitialized storage pointers. • Arithmetic accuracy. • Design Logic. • Cross-function race conditions. • Safe Zeppelin module. • Fallback function security. • Overpowered functions / Owner privileges

AUDIT OVERVIEW

1. Zero-Price / Underpriced Token Minting [Fixed ✅]

Location: simulateBuy() and buy() [Token contract]

Issue: Integer divisions collapse small values to zero, so attackers can mint tokens for free or near-free.

Recommendation: Use fixed-point arithmetic (for example Q112), scale intermediate multipliers, and add require(value > 0).

Fix: Fixed-point math (Q18) is used throughout to keep precision high and avoid tiny rounding errors.

Any tokens > 0 that would lead to value == 0 is explicitly forbidden via revert ZeroValueTransaction().

2. Incorrect Price Calculation Precision [Fixed ✅]

Location: Token k, i constants & simulateBuy/sell

Issue: k = 37_5 evaluates to 375 (not 37.5) and you divide by 0.0001 ether (1e14) the combination causes major precision loss and likely wrong prices. The first 400 tokens (with 18 decimal places) can be claimed for free by anyone, as long as the deployer hasn't set an initial buy price. Subsequent purchases cost close to zero to mint new tokens, depending on how the values of k and i are configured.

Recommendation: Use clear fixed-point notation (e.g. k = 375e15; i = 8e22; for Q18), multiply before dividing, or adopt a standard fixed-point library. Document the formula.
Fix: By defining k and i in clear Q18 form and doing multiplies before divides (plus the zero-value check), dev team eliminated the 'first 400 tokens free' / 'near-zero subsequent buys' vulnerability.

3. Forced-BNB donation breaks the curve cause permanent buy DoS and fund redistribution [Fixed ✅]

Location: simulateBuy, simulateSell

Issue: All buy/sell pricing uses the contract's live BNB balance:

Buy Sim:

```
xBefore = VIRTUAL_BNB + currentBalance xAfter  = k / yAfter value   = xAfter - xBefore
require(xAfter > xBefore)
```

where currentBalance is address(this).balance (less msg.value).

Sell sim uses xBefore = VIRTUAL_BNB + address(this).balance.

Because anyone can force BNB into a contract (for example selfdestruct/coinbase refunds), address(this).balance is not an invariant you can trust. An attacker can push enough BNB so that, for any tokens, xAfter <= xBefore, triggering InvalidBondingCurveCalculation() and bricking all buys forever. With the chosen constants:

- k = VIRTUAL_BNB * VIRTUAL_TOKENS
- VIRTUAL_BNB = 3.6 ether
- VIRTUAL_TOKENS = 960,000,000 ether
- Minimum possible yAfter before listing is VIRTUAL_TOKENS - MAX_SELL = 160,000,000
- ether Hence maximum possible `xAfter = k / yAfter_min = 21.6 ether**

Therefore, if an attacker forces > 18.0 BNB into the contract (so xBefore = 3.6 + balance ≥ 21.6), all buys revert for the rest of the lifecycle (you cant progress to MAX_SELL, so you'll never list a large forced balance increases sell payouts (value = xBefore - xAfter), meaning holders (or the attacker if they hold tokens) can siphon out the forced BNB by selling into the curve.

If the attacker times this early (when supply is small), there may not be enough tokens to drain but the buy path is already bricked, so no one can acquire more to fix it.

Recommendation: Never use address(this).balance in bonding curve math. Track an internal reserve accounting instead: Introduce uint256 internalReserve On buy: compute value from curve; update internalReserve += value. On sell: compute value; update internalReserve -= value. Use internalReserve (not address(this).balance) in all x/y/k calculations. Add a receive() that reverts to prevent accidental sends (note: can't prevent selfdestruct, so internal accounting is still mandatory). Optionally add an owner-only skim() to move unexpected BNB to a treasury address without affecting reserve math.
Fix: Introduced an internal accounting reserve and all curve math now uses internalReserve instead of address(this).balance. Forced BNB (via selfdestruct, etc.) can no longer brick buys or inflate sells, it doesn't enter the pricing state (internalReserve). This removes the DoS and the prior economic manipulation vector.

1. Unrestricted list() Exposure [Acknowledge]

Location: Token.list()

Issue: Public visibility lets any EOA trigger listing once caps are reached.

Recommendation: Restrict to owner or internal call.

2. Reentrancy Vulnerability in sell() Function [Fixed ✅]

Location: Token.sell()

Issue: The sell() function performs a transfer before updating state (calling _burn)

While the state is updated in the correct order for the token balance, the external call happens before fee processing

Recommendation: Follow checks-effects-interactions pattern strictly

Move the _burn call before the ETH transfer

Consider using OpenZeppelin ReentrancyGuard

Fix: Now the contract burn before sending funds, so the classic reentrancy-through-unsynced-state is gone.

LOW

No low issues were found.

INFORMATIONAL

1. ETH Balance Handling Issues

Issue: The contract doesn't properly handle leftover ETH in several scenarios:

In buy(), excess ETH sent isn't returned

In list(), the entire balance is converted to WBNB without considering pending sells

Recommendation: Add refund mechanism for excess ETH in buy()

Separate ETH reserves for buy/sell liquidity vs listing liquidity

Consider adding a withdrawal function for contract owner

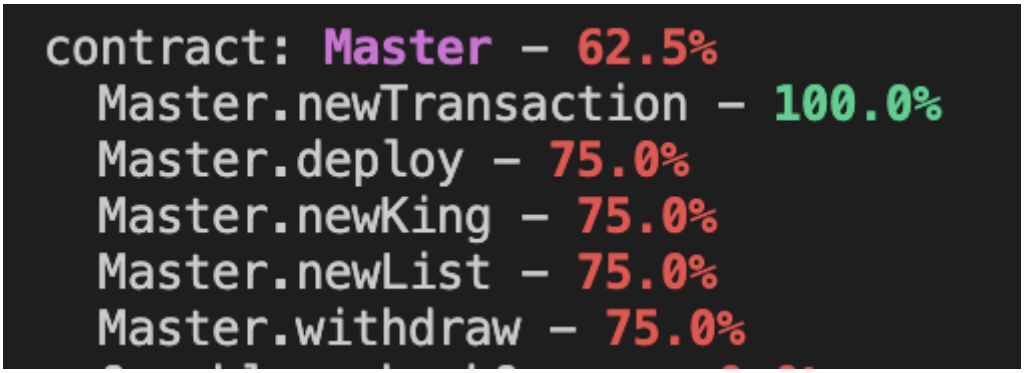
2. KING_SELL Trigger Logic

Issue: The master.newKing() is called when getTokenSold() >= KING_SELL, which means it could be triggered multiple times if tokens are sold back after reaching the threshold.

Recommendation: Track whether the king event has already been triggered or make it trigger only when crossing the threshold upwards

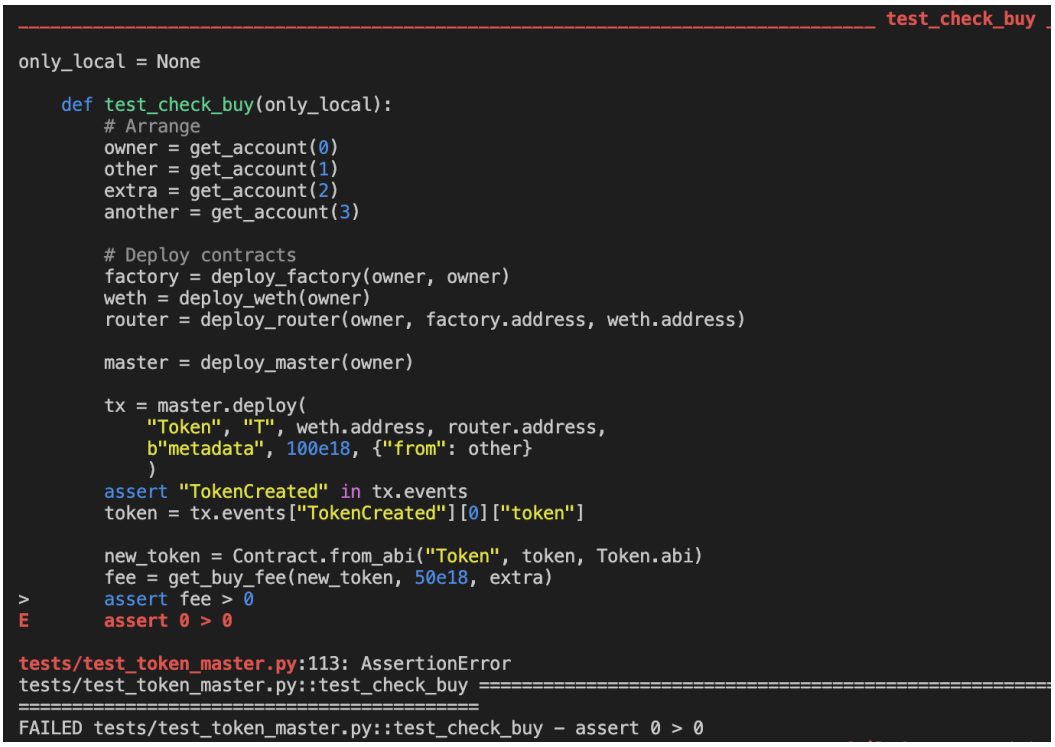
Testing coverage

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



```
tests/test_token_master.py::test_buy RUNNING
Transaction sent: 0x630175e2b8028df55fadb3f2e43c4b8fd480cfcbd10f45155fb85cb188b4a285
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  UniswapV2Factory.constructor confirmed   Block: 1   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

Transaction sent: 0x1f89adc4fa9d3ed1c5cf06121dbc32c29e891456f0c7ad7e9015147d80d0ede7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  WETH9.constructor confirmed   Block: 2   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0x602C71e4DAC47a042Ee7f46E0aee17F94A3bA0B6

Transaction sent: 0xf99c34447c2dec1afc53e0e268dee0ab9253f622233d4b39a4c948d17ad8c81a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  UniswapV2Router02.constructor confirmed   Block: 3   Gas used: 3895430 (32.46%)
  UniswapV2Router02 deployed at: 0xE7eD6747FaC5360f88a2EFC03E00d25789F69291

Transaction sent: 0xfefa3155ee6cb550bd56491da9d0f80cdd6764c9cccc5484f7cfa59d2a4ee0cc
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  Master.constructor confirmed   Block: 4   Gas used: 2194663 (18.29%)
  Master deployed at: 0x6951b5Bd815043E3F842c1b026b0Fa888Cc2DD85

Transaction sent: 0xa90d5113839dbac62b30b0dedbcc323a56ef4519f8727cd8f54a8d25030b85a4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Token.constructor confirmed   Block: 5   Gas used: 1519080 (12.66%)
  Token deployed at: 0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9

Transaction sent: 0x64a05b719c283de97573700e2effd1691833a8a939d9602de23a637b937e494b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Token.buy confirmed (Not token)   Block: 6   Gas used: 78987 (0.66%)

Transaction sent: 0x2f8161ece5c5f6627b8298decc31e5ab33387eb19b8a0b80e3743f7c6d5b25dc
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Master.deploy confirmed   Block: 7   Gas used: 1505130 (12.54%)

Transaction sent: 0x1bb5fc1b48fc733cf182f42cbc4f7e22417667d7fadcfd6ab0979796e0cb968d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Transaction confirmed (reverted)   Block: 8   Gas used: 27692 (0.23%)

Transaction sent: 0xfe55a66726f4cdeda2578409a01912f07f93fbb89ec4c78ddda9631675465b44
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Transaction confirmed   Block: 9   Gas used: 60398 (0.50%)

Transaction sent: 0x6d43f97dd4f766eb18d408a8a7611459d0dc91b3e0c15cf3cfd4da71f344ad66
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Transaction confirmed   Block: 10   Gas used: 75386 (0.63%)

Transaction sent: 0x8cce4608a2fa7c05123feb7b944dc0c3282c2b19e69a9a715dcf3fb295b7f735
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  Transaction confirmed   Block: 11   Gas used: 75386 (0.63%)

Transaction sent: 0xea4d5b2e7092c755a2cbffb9ceda5b10d651849428529770ba7624e7d4b1e671
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  Transaction confirmed (reverted)   Block: 12   Gas used: 28658 (0.24%)

Transaction sent: 0x6826be75dbc464941354c274d6e12227b90488a11b5ff12f545dea3f2d90f71a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Transaction confirmed (reverted)   Block: 13   Gas used: 22155 (0.18%)

Transaction sent: 0xf7ea1967dca00b79f676a1ec6b134f5ec811a5ac36219b40fcb76a61dafb950b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  Transaction confirmed   Block: 14   Gas used: 2285527 (19.05%)

Transaction sent: 0xb658cb92c65e384cbc5cd88ce8bacddcfdaa418d1f717d7c13fa2051d547f2f7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  Transaction confirmed (reverted)   Block: 15   Gas used: 22923 (0.19%)

Transaction sent: 0x6975160929bb3f266b18784ecbac304a279d27cbb434636cfae6d26a6839eefe
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  Transaction confirmed (reverted)   Block: 16   Gas used: 22155 (0.18%)

Transaction sent: 0x6b4c54b8edaf4895f9d152eb6a43f020d662bd96a731ca0acbeefe7e6968a39f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Transaction confirmed (reverted)   Block: 17   Gas used: 22405 (0.19%)

tests/test_token_master.py::test_buy PASSED
```

```
tests/test_token_master.py::test_check_buy RUNNING
Transaction sent: 0x32b8acb9e864a45fae14f659be5bceb17f87987c8444d0c2979d7313625b571c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  UniswapV2Factory.constructor confirmed   Block: 18   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3

Transaction sent: 0xc3b1c88f75c07713f631d45d8ef7aa9d6e0278aca70693d0395072c3e52d7db1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  WETH9.constructor confirmed   Block: 19   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0x9E4c14403d7d9A8A782044E86a93CAE09D7B2ac9

Transaction sent: 0x0068811484037b75df39cdae4365924fbb3cedeb3168dcc61422d7433b123f57
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  UniswapV2Router02.constructor confirmed   Block: 20   Gas used: 3895430 (32.46%)
  UniswapV2Router02 deployed at: 0xcCB53c9429d32594F404d01fbe9E65ED1DCda8D9

Transaction sent: 0x98feafd9ffd7e584121b83ba3c3e9e4456cfaa9d13802771ffacf55f2bc23a0c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  Master.constructor confirmed   Block: 21   Gas used: 2194663 (18.29%)
  Master deployed at: 0x420b1099B9eF5baba6D92029594eF45E19A04A4A

Transaction sent: 0x7d8808c998a854f623004a4f4f8ec62b7786ba0510bb762f45da05722ade3ee2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  Master.deploy confirmed   Block: 22   Gas used: 1505142 (12.54%)

tests/test_token_master.py::test_check_buy FAILED
tests/test_token_master.py::test_sell RUNNING
Transaction sent: 0x075d8e5602d809184df1419c32506d903f264706abbe9ffceaa7696ad9175a70
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  UniswapV2Factory.constructor confirmed   Block: 23   Gas used: 2412742 (20.11%)
  UniswapV2Factory deployed at: 0xa3B53dDCd2E3fC28e8E130288F2aBD8d5EE37472

Transaction sent: 0x7c9de625c8d4938854f903bf5f83ecfc9a6aeb8324e33957f644e6ebefef6cfe
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  WETH9.constructor confirmed   Block: 24   Gas used: 476546 (3.97%)
  WETH9 deployed at: 0xb6286fAFd0451320ad6A8143089b216C2152c025

Transaction sent: 0x0d693be4590e141c8c3ab00f57b1325d9d48d17d495e25eae24edaa1041acc01
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  UniswapV2Router02.constructor confirmed   Block: 25   Gas used: 3895430 (32.46%)
  UniswapV2Router02 deployed at: 0x7a3d735ee6873f17Dbdcab1d51B604928dc10d92

Transaction sent: 0x9115d48e93f63cf9209b34c22aaf2e1217d5f3f6529bffff092bc84f9c4f3f06
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  Master.constructor confirmed   Block: 26   Gas used: 2194663 (18.29%)
  Master deployed at: 0x2c15A315610Bfa5248E4CbCbd693320e9D8E03Cc

Transaction sent: 0xf1de4821e96325316d42caee0463413d4a392c43c4b0f9bfed481767361f5b7c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  Master.deploy confirmed   Block: 27   Gas used: 1505142 (12.54%)

Transaction sent: 0x8eaf755c940226979c502f7db71641e7efad509fc13b5c251295e1dfd9ad3b0c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  Transaction confirmed (reverted)   Block: 28   Gas used: 23290 (0.19%)

Transaction sent: 0x0b409e8ce15ade6648cf0577dc422f906f98f8a71edb2ee7c8f0218c0663299f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  Transaction confirmed   Block: 29   Gas used: 75386 (0.63%)

Transaction sent: 0xbd19c802eaecf5dd0e971ae97b80b6d9544b3b210b75027eeaa758f0babb621d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  Transaction confirmed   Block: 30   Gas used: 67350 (0.56%)

tests/test_token_master.py::test_sell PASSED
```


test_check_buy
only_local = None
def test_check_buy(only_local):
# Arrange
owner = get_account(0)
other = get_account(1)
extra = get_account(2)
another = get_account(3)
# Deploy contracts
factory = deploy_factory(owner, owner)
weth = deploy_weth(owner)
router = deploy_router(owner, factory.address, weth.address)
master = deploy_master(owner)
tx = master.deploy(
"Token", "T", weth.address, router.address,
b"metadata", 100e18, {"from": other}
)
assert "TokenCreated" in tx.events
token = tx.events["TokenCreated"][0]["token"]
new_token = Contract.from_abi("Token", token, Token.abi)
fee = get_buy_fee(new_token, 50e18, extra)
> assert fee > 0
E assert 0 > 0
tests/test_token_master.py:113: AssertionError
tests/test_token_master.py::test_check_buy
FAILED tests/test_token_master.py::test_check_buy - assert 0 > 0

Annexes

Testing code:

```
from brownie import (
    reverts,
    Token
)

from brownie.network.contract import Contract

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
)

from scripts.deploy import (
    deploy_weth,
    deploy_router,
    deploy_factory,
    deploy_token,
    deploy_master
)

def test_buy(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    another = get_account(3)

    # Deploy contracts
    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)
```

```
master = deploy_master(owner)
token = deploy_token(owner, master.address, weth.address, router.address)

with reverts("Not token"):
    token.buy(1000, other, {"from": extra})

tx = master.deploy(
    "Token", "T", weth.address, router.address,
    b"metadata", 400e18, {"from": other}
    )
assert "TokenCreated" in tx.events
token = tx.events["TokenCreated"][0]["token"]

new_token = Contract.from_abi("Token", token, Token.abi)

with reverts("InsufficientFunds: "):
    new_token.buy(50e18, other, {"from": extra, "value": 0})

fee = get_buy_fee(new_token, 50e18, extra)
tx = new_token.buy(50e18, other, {"from": extra, "value": fee})
assert tx.events['Transfer'][0]['from'] == ZERO_ADDRESS
assert tx.events['Transfer'][0]['to'] == other
assert tx.events['Transfer'][0]['value'] == 50e18

fee = get_buy_fee(new_token, 1000e18, extra)
tx = new_token.buy(1000e18, extra, {"from": other, "value": fee})
assert tx.events['Transfer'][0]['from'] == ZERO_ADDRESS
assert tx.events['Transfer'][0]['to'] == extra
assert tx.events['Transfer'][0]['value'] == 1000e18

fee = get_buy_fee(new_token, 10e18, extra)
tx = new_token.buy(10e18, another, {"from": another, "value": fee})
assert tx.events['Transfer'][0]['from'] == ZERO_ADDRESS
assert tx.events['Transfer'][0]['to'] == another
```

```
    assert tx.events['Transfer'][0]['value'] == 10e18


    fee = get_buy_fee(new_token, 80000000e18, extra)
    with reverts("MaxSupplyReached: "):
        new_token.buy(80000000e18, another, {"from": another, "value": fee})


    with reverts("NotReady: "):
        new_token.list({"from": other})


    fee = get_buy_fee(new_token, 80000000e18 - 1460e18, extra)
    tx = new_token.buy(80000000e18 - 1460e18, another, {"from": another, "value":
fee})
    assert tx.events['TokenListed'][0]['token'] == token


    with reverts("Untradable: "):
        new_token.buy(100e18, another, {"from": another, "value": fee})


    with reverts("NotReady: "):
        new_token.list({"from": other})


    with reverts("Untradable: "):
        new_token.sell(100e18, {"from": other})



def test_check_buy(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    another = get_account(3)


    # Deploy contracts
    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)
```

```
    master = deploy_master(owner)

    tx = master.deploy(
        "Token", "T", weth.address, router.address,
        b"metadata", 100e18, {"from": other}
        )
    assert "TokenCreated" in tx.events
    token = tx.events["TokenCreated"][0]["token"]

    new_token = Contract.from_abi("Token", token, Token.abi)
    fee = get_buy_fee(new_token, 50e18, extra)
    assert fee > 0
    tx = new_token.buy(50e18, extra, {"from": extra, "value": fee})
    assert tx.events['Transfer'][0]['from'] == ZERO_ADDRESS
    assert tx.events['Transfer'][0]['to'] == extra
    assert tx.events['Transfer'][0]['value'] == 50e18


def test_sell(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    another = get_account(3)

    # Deploy contracts
    factory = deploy_factory(owner, owner)
    weth = deploy_weth(owner)
    router = deploy_router(owner, factory.address, weth.address)

    master = deploy_master(owner)

    tx = master.deploy(
        "Token", "T", weth.address, router.address,
```

```
        b"metadata", 400e18, {"from": other}
        )
    assert "TokenCreated" in tx.events
    token = tx.events["TokenCreated"][0]["token"]

    new_token = Contract.from_abi("Token", token, Token.abi)

    with reverts("InsufficientTokens: "):
        new_token.sell(10e18, {"from": extra})

    fee = get_buy_fee(new_token, 100e18, extra)
    tx = new_token.buy(100e18, extra, {"from": extra, "value": fee})
    assert tx.events['Transfer'][0]['from'] == ZERO_ADDRESS
    assert tx.events['Transfer'][0]['to'] == extra
    assert tx.events['Transfer'][0]['value'] == 100e18

    tx =  new_token.sell(10e18, {"from": extra})
    assert tx.events['Transfer'][0]['from'] == extra
    assert tx.events['Transfer'][0]['to'] == ZERO_ADDRESS
    assert tx.events['Transfer'][0]['value'] == 10e18



def get_buy_fee(token, amount, account):
    res = token.simulateBuy(amount, {"from": account})
    fee = (res[0] + (res[0] * 100) / 10000)
    return fee

def get_sell_fee(token, amount, account):
    res = token.simulateSell(amount, {"from": account})
    fee = ((res[0] * 100) / 10000)
    return fee
```

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	3					3
Medium	2			1		1
Low	2					
Informational	0					

Assessment Results

Score Results

Review	Score
Global Score	**85/100**
Assure KYC	Not completed
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit PASS

Following our comprehensive security audit of the token contract for the OnlyPump project, the project did meet the necessary criteria required to pass the security audit.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial OnlyPump in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment OnlyPump, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment OnlyPumps provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any OnlyPump reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The OnlyPump may access, and depend upon, multiple layers of third parties.