Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Prophet

Date: 19/01/2024

Audit Status: PASS

Audit Edition: Advanced

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🟡 Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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 **Well Secured**

Insecure	Poorly Secured	Secured	**Well Secured**

Scope

Target Code And Revision

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

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	ProphetRouterV1.sol [SHA256] - 313667712bfd747287dcff604cfe200767f97604 ff40157e5278289ab1955a25
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

HIGH

No high severity issues were found.

MEDIUM

1. Unlimited fees

Contract: ProphetRouterV1.sol

Functions: ProphetBuy(), ProphetMaxBuy(), ProphetSell(), ProphetSmartSell(), swapTokensSupportingFeeOnTransferTokensForExactETH()

Issue: The fees do not have any set limit, allowing, in the case of, for example, a leak of the admin private key, to convert the contract into a honey pot if the limit is set to 100%.

Fix: We recommend adding a limit to 10-15%.

LOW

No Low severity issues were found.

INFORMATIONAL

No informational issues were found.

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.*

Prophet contracts tests

```
contract: ProphetRouterV1 — 66.7%
  ProphetRouterV1.ProphetBuy — 100.0%
  ProphetRouterV1.getPathForTokenToToken — 100.0%
  ProphetRouterV1.setTokenToEther — 100.0%
  ProphetRouterV1.ProphetSmartSell — 91.7%
  TransferHelper.safeTransferFrom — 83.3%
  UniswapV2Library.getAmountIn — 83.3%
  UniswapV2Library.getAmountOut — 83.3%
  ProphetRouterV1.ProphetSell — 75.0%
  ProphetRouterV1._swapSupportingFeeOnTransferTokens — 75.0%
  SafeMathS.add — 75.0%
  SafeMathS.mul — 75.0%
  SafeMathS.sub — 75.0%
  TransferHelper.safeTransferETH — 75.0%
  UniswapV2Library.getAmountsIn — 75.0%
  UniswapV2Library.getReserves — 75.0%
  UniswapV2Library.sortTokens — 75.0%
  ProphetRouterV1.ProphetMaxBuy — 73.5%
  ProphetRouterV1.swapTokensSupportingFeeOnTransferTokensForExactETH — 62.5%
  ProphetRouterV1.transferOwnership — 0.0%
  TransferHelper.safeTransfer — 0.0%
  UniswapV2Library.getAmountsOut — 0.0%
  UniswapV2Library.quote — 0.0%
```

```
tests/test_prophet_router.py::test_prophet_buy RUNNING
Transaction sent: 0x072ab49b2d5a2e70fb44f1dfa7a104e9683e1f4feaebfe218e2f4ebf8a1b4d93
Transaction sent: 0x2779d0b356a5de7ad4b75f0829cce1f588e380b10401259015fe3fea07774f9d
Transaction sent: 0x0698eb546eda2286937dc8a082da6a8a0196aee7e95b22c3608f012fd38fe8f3
tests/test_prophet_router.py::test_prophet_buy PASSED
tests/test_prophet_router.py::test_prophet_smart_sell RUNNING
Transaction sent: 0x350d9d0f59a33001c9751bdb431ab070568ccae6b0e2dff33864c734bcc50f89
Transaction sent: 0xb29db853da5aa5ae18fee8513a03b59ede8780eb3e09019cb4d70f44d93a76ea
Transaction sent: 0x5c630530ca63d94d2d121c02558abba930a04759f6144a53cf21ad6ab7e3e4b3
Transaction sent: 0x6329baef7bb0babdbffc3b7e8d3d1fff677d18aacdd8e90e9004c21f3d50eca9
tests/test_prophet_router.py::test_prophet_smart_sell PASSED
tests/test_prophet_router.py::test_prophet_sell RUNNING
Transaction sent: 0x5a5585f170b9d9bb6b175d63ee26d1354b560a36901649e9b1d7fc1f6d43dad8
Transaction sent: 0xb844dbc7271dbeb022e5333ddf475f4c0a246cf833d29a586542d3a060f38502
Transaction sent: 0xac6e216507fb98cd83f0fcb494cce867dd6501a86b36433a3630e5979255d1f9
tests/test_prophet_router.py::test_prophet_sell PASSED
tests/test_prophet_router.py::test_prophet_max_buy RUNNING
Transaction sent: 0x129218b0fb65db5465ec805f024ff30ee36148420005b63c17f55cc1db1ee654
Transaction sent: 0x1d0ce65ebafb3ea99b44bf8024b97cfc89318311818c137c696439cefd930198
Transaction sent: 0xc385eb37e666b4b9161adcfdf756f5847818c69f3144d4d0f17cf23f5d6b0894
Transaction sent: 0x594531c174f09cf1e621975e0cd334fa6f0022936286c8e9b632e29a30aecc0b
Transaction sent: 0xd5f0e3e3965c6081056b929b416caca9047f2fc70e223d05e5ec37748102247c
Transaction sent: 0xf682e091ea431fa1e78554db4fc88abfc6b494c82c819443e382ef50eb407e95
tests/test_prophet_router.py::test_prophet_max_buy PASSED

======================================================================= 4 passed in 4.74s
```

Annexes

Testing code:

```python
Test_prophet_router.py
from brownie import (
    reverts,
)

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

from scripts.deploy import (
    deploy_factory,
    deploy_router,
    deploy_liquidity,
    deploy_mock_weth,
    deploy_token,
    deploy_prophet
)

def test_prophet_buy(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    fee_admin = get_account(3)

    weth = deploy_mock_weth(owner)
    token = deploy_token(owner)
    factory = deploy_factory(owner, fee_admin)
    router = deploy_router(owner, factory.address, weth.address)
    liquidity = deploy_liquidity(owner, factory.address, router.address, weth.address,
token.address)
    prophet = deploy_prophet(owner, factory.address, weth.address)

    # assert
    deadline = get_timestamp()
    with reverts("PropherRouter: INVALID_FEE_AMOUNT"):
        prophet.ProphetBuy(1e18, token.address, other, deadline, 1, {"from": extra})

    with reverts():
        prophet.ProphetBuy(1e18, token.address, other, deadline, 1, {"from": extra,
```

```python
    "value": 1e18})

    tx = factory.createPair(weth.address, token.address, {"from": owner})
    pair_addr = tx.events['PairCreated'][0]['pair']

    with reverts("UniswapV2Library: INSUFFICIENT_LIQUIDITY"):
        prophet.ProphetBuy(1e18, token.address, other, deadline, 1, {"from": extra,
    "value": 1e18})

    weth.deposit({"from": owner, "value": 10e18})
    weth.approve(liquidity.address, 100e18, {"from": owner})
    token.approve(liquidity.address, 100e18, {"from": owner})
    # Add liquidity
    liquidity.addLiquidity(weth.address, token.address, 5e18, 5e18, {"from": owner})
    # Buy
    tx = prophet.ProphetBuy(1e17, token.address, extra, deadline, 1, {"from": extra,
    "value": 1e18})
    assert tx.events["Deposit"] is not None
    assert tx.events["Transfer"] is not None
    assert tx.events["Sync"] is not None
    assert tx.events["ProphetFee"] is not None

def test_prophet_smart_sell(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    fee_admin = get_account(3)
    account = get_account(4)

    weth = deploy_mock_weth(owner)
    token = deploy_token(owner)
    factory = deploy_factory(owner, fee_admin)
    router = deploy_router(owner, factory.address, weth.address)
    liquidity = deploy_liquidity(owner, factory.address, router.address, weth.address,
    token.address)
    prophet = deploy_prophet(owner, factory.address, weth.address)

    # assert
    deadline = get_timestamp()
    with reverts("PropherRouter: FEE_AMOUNT"):
        prophet.ProphetSmartSell(1e18, 2e18, token.address, deadline, 0, {"from":
    extra})

    tx = factory.createPair(weth.address, token.address, {"from": owner})
    pair_addr = tx.events['PairCreated'][0]['pair']

    with reverts("UniswapV2Library: INSUFFICIENT_LIQUIDITY"):
        prophet.ProphetSmartSell(1e18, 2e18, token.address, deadline, 1, {"from":
```

```python
extra})

    weth.deposit({"from": owner, "value": 10e18})
    weth.approve(liquidity.address, 100e18, {"from": owner})
    token.approve(liquidity.address, 100e18, {"from": owner})
    # Add liquidity
    liquidity.addLiquidity(weth.address, token.address, 5e18, 5e18, {"from": owner})

    with reverts():
        prophet.ProphetSmartSell(1e18, 2e18, token.address, deadline, 1, {"from":
extra})

    # Send approval
    weth.approve(prophet.address, 100e18, {"from": extra})
    token.approve(prophet.address, 100e18, {"from": extra})
    weth.transfer(extra, 5e18, {"from": owner})
    token.transfer(extra, 5e18, {"from": owner})

    with reverts("PropherRouter: EXCESSIVE_INPUT_AMOUNT"):
        prophet.ProphetSmartSell(1e18, 2e18, token.address, deadline, 10000, {"from":
extra})

    tx = prophet.ProphetSmartSell(1e18, 2e18, token.address, deadline, 1000, {"from":
extra})
    assert tx.events["Approval"] is not None
    assert tx.events["Transfer"] is not None
    assert tx.events["Sync"] is not None
    assert tx.events["ProphetFee"] is not None

def test_prophet_sell(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    fee_admin = get_account(3)

    weth = deploy_mock_weth(owner)
    token = deploy_token(owner)
    factory = deploy_factory(owner, fee_admin)
    router = deploy_router(owner, factory.address, weth.address)
    liquidity = deploy_liquidity(owner, factory.address, router.address, weth.address,
token.address)
    prophet = deploy_prophet(owner, factory.address, weth.address)

    # assert
    deadline = get_timestamp()
    with reverts("TransferHelper::transferFrom: transferFrom failed"):
        prophet.ProphetSell(1e18, 2e18, token.address, deadline, 0, {"from": extra})
```

```python
    tx = factory.createPair(weth.address, token.address, {"from": owner})
    pair_addr = tx.events['PairCreated'][0]['pair']

    token.approve(prophet.address, 10e18, {"from": owner})
    weth.deposit({"from": owner, "value": 10e18})
    weth.approve(prophet.address, 10e18, {"from": owner})
    with reverts("UniswapV2Library: INSUFFICIENT_LIQUIDITY"):
        prophet.ProphetSell(1e18, 2e18, token.address, deadline, 0, {"from": owner})

    weth.deposit({"from": owner, "value": 10e18})
    weth.approve(liquidity.address, 100e18, {"from": owner})
    token.approve(liquidity.address, 100e18, {"from": owner})
    # Add liquidity
    liquidity.addLiquidity(weth.address, token.address, 5e18, 5e18, {"from": owner})

    with reverts():
        prophet.ProphetSell(1e18, 1e18, token.address, deadline, 0, {"from": extra})

    # Send approval
    weth.approve(prophet.address, 100e18, {"from": extra})
    token.approve(prophet.address, 100e18, {"from": extra})
    token.transfer(extra, 5e18, {"from": owner})

    tx = prophet.ProphetSell(2e18, 1e18, token.address, deadline, 100, {"from":
extra})
    assert tx.events["Approval"] is not None
    assert tx.events["Transfer"] is not None
    assert tx.events["Sync"] is not None
    assert tx.events["ProphetFee"] is not None

def test_prophet_max_buy(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    fee_admin = get_account(3)

    weth = deploy_mock_weth(owner)
    token = deploy_token(owner)
    factory = deploy_factory(owner, fee_admin)
    router = deploy_router(owner, factory.address, weth.address)
    liquidity = deploy_liquidity(owner, factory.address, router.address, weth.address,
token.address)
    prophet = deploy_prophet(owner, factory.address, weth.address)

    # assert
    deadline = get_timestamp()
    with reverts("PropherRouter: EXCEEDED_ETHER_LIMIT"):
        prophet.ProphetMaxBuy(1e18, token.address, extra, deadline, 0, {"from": extra,
```

```python
    "value": 1e18})

    prophet.ProphetMaxBuy(1e17, token.address, extra, deadline, 0, {"from": extra,
"value": 1e17})

    # send token
    with reverts("PropherRouter: NOT OWNER"):
        prophet.setTokenToEther(token.address, 1e18, {"from": extra})
    with reverts("PropherRouter: ZERO_VALUE"):
        prophet.setTokenToEther(token.address, 0, {"from": owner})
    prophet.setTokenToEther(token.address, 1e18, {"from": owner})

    with reverts("PropherRouter: INVALID_FEE_AMOUNT"):
        prophet.ProphetMaxBuy(1e17, token.address, extra, deadline, 0, {"from": extra,
"value": 1e17})

    with reverts():
        prophet.ProphetMaxBuy(1e17, token.address, extra, deadline, 1, {"from": extra,
"value": 1e17})
    prophet.setTokenToEther(token.address, 1e16, {"from": owner})
    with reverts():
        prophet.ProphetMaxBuy(1e17, token.address, extra, deadline, 1, {"from": extra,
"value": 1e17})

    weth.deposit({"from": owner, "value": 10e18})
    weth.approve(liquidity.address, 100e18, {"from": owner})
    token.approve(liquidity.address, 100e18, {"from": owner})
    # Add liquidity
    liquidity.addLiquidity(weth.address, token.address, 5e18, 5e18, {"from": owner})

    prophet.setTokenToEther(token.address, 5e18, {"from": owner})
    prophet.ProphetMaxBuy(1e17, token.address, extra, deadline, 1000, {"from": extra,
"value": 1e18/2})

    prophet.setTokenToEther(token.address, 1e17, {"from": owner})
    prophet.ProphetMaxBuy(1e16, token.address, extra, deadline, 1, {"from": extra,
"value": 2e17})


    prophet_v2 = deploy_prophet(owner, factory.address, weth.address)
    prophet_v2.ProphetMaxBuy(1e17, token.address, extra, deadline, 1, {"from": extra,
"value": 1e17})
```

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Resolved
🔴 High	0				
🟠 Medium	1				1
🟡 Low	0				
🟢 Informational	0				

Assessment Results

Score Results

Review	Score
Global Score	**95/100**
Assure KYC	Completed https://assuredefi.com/projects/prophetbots
Audit Score	90/100

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.

<u>Audit Pass</u>

Following our comprehensive security audit of the Solidity project Prophet, we inform you that the project has met the necessary security standards. The audit identified a medium-level issue that could potentially deplete user funds in the event of a key leak, or if the team incorrectly sets an excessively high fee (100%) but the team applied a fix adding a 10% limit.

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 advice 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 advice, 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 services 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 services, 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 services may access, and depend upon, multiple layers of third parties.

ASSURE DEFI®
THE VERIFICATION **GOLD STANDARD**