Assure DeFi®

THE VERIFICATION **GOLD STANDARD**

VERIFIED BY ASSURE DEFI® INTEGRITY • TRUST • CREDIBILITY

Security Assessment

Modus

Date: 21/01/2024

Audit Status: Fail

Audit Edition: Advanced

ASSURE DEFI®
THE VERIFICATION **GOLD STANDARD**

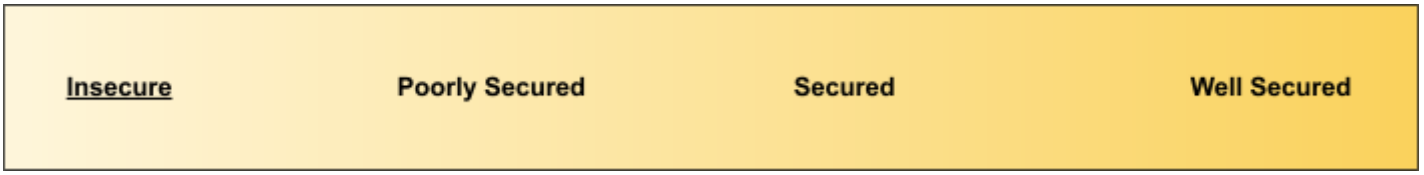
Risk Analysis

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Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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



Scope

Target Code And Revision

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

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	File hashes [SHA256]: KYCModule.sol - *ada0ac93f1c43cc323c81c3a0436c6d429c8b62c953310d71bc899be2e4b0eca* ModusProxyFactory.sol - *346c583751ff0a0dd580420ad6000d7852eafa8cd4fdc37d32e67a381cef2b33* ModusRegistry.sol - *98342c29dd1a2e89174e4718a2067b7d7b43ab59815a4da9444db751e1b477ce* PXTStaking.sol - 00120e862b4b38c50b8f6bef34e0999ada48b3dae4f274163a6851305bf7de8c StableCoinPXTSwapv2.sol - aae1dc9772e9380e31a6e32b7bb7ed6e139e46f0cdcda38f1056d4f587c17344 TokenStableCoinSwap.sol - d74f8efbdbcc0c1c078526a4273f72e8a9f5b12e5589e1e999c74f40f1e2afa9
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

1. Reentrancy in stake/unstake

Contract: PXTStaking

Functions: stake(), unstake()

Issue: Staking and unstaking functions are potentially insecure before the first invocation is fully executed, potentially draining the contract's funds or manipulating staking balances.

Fix: Add nonReentrant guard modifier.

MEDIUM

1. Conversion rate not limited

Contract: TokenStableCoinSwap

Functions: initialize()

Issue: The lack of a conversion limit can lead to market manipulation and slippage issues.

Fix: We recommended implementing a maximum conversion limit per transaction.

2. Missing validation check for Staked Amount

Contract: PXTstaking

Functions: adminUnstakeForLostwallet()

Issue: Staked amount balance [>0] is not checked before critical operations.

Fix: Add a require(investorDetails[oldWallet].stakedAmount > 0).

1. Zero address Validation Missing

Contract: TokenStableCoinSwap, StableCoinPXTSwapv2, PXTStaking, ModusRegistry, KYCModule

Functions:

TokenStableCoinSwap:

Initialize(): tokenAddress can be 0 address.

StableCoinPXTSwapv2:

setPremiumInvestor(): Investor can be 0 address.

setPremiumInvestorsList(): Investor can be 0 address.

PXTStaking():

adminUnstakeForLostwallet(): New Wallet can be 0 address.

ModusRegistry():

constructor(): _modusTreasuryWallet, _admin and _lostWalletAdmin can be 0 address.

setImplementationContracts(): All implementations can be 0 address.

changeStableCoinPXTSwap(): _stablecoinpxtSwap can be 0 address.

setStableCoin(): _stableCoin can be 0 address.

KYCModule:

setKYCStatus(): Investors can have 0 address.

Issue: During the audit we detected that multiple variables can be set to 0 address leading to potentially lost funds or unintended behaviors.

Fix: Implement checks to validate that the address is not the zero address (0x0) in the affected functions/contracts.

2. Unchecked Project Funding Cap

Contract: StableCoinPXTSwapv2

Functions: initialize()

Issue: There is no max or min _projectcap check performed during the initialization.

Fix: Add max and min _projectcap.

3. Hardcoded address

Contract: StableCoinPXTSwapv2

Functions: bridgeToEth()

Issue: The ‘ambBridge’ address is hardcoded restricting the flexibility and upgradability of the contract, as any change in these addresses would require a full redeployment of the contract

Fix: Create set and get ambBridgeAddress.

4. Unchecked array size

Contract: StableCoinPXTSwapv2

Functions: setPremiumInvestorsList()

Issue: Excessively large arrays can cause the contract to run out of gas during operations that iterate over them, rendering these functions unusable.

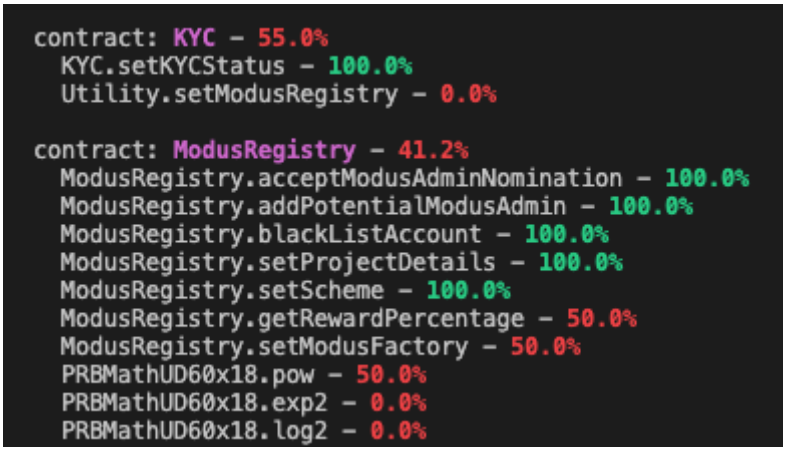
Fix: Define an array limit for setPremiumInvestorsList().

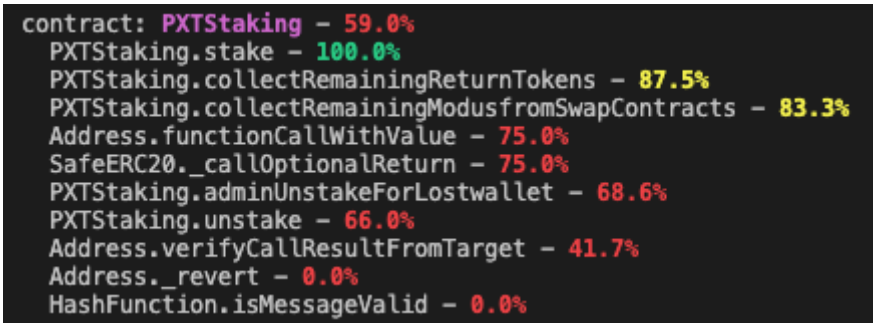
No Informational severity issues were found.

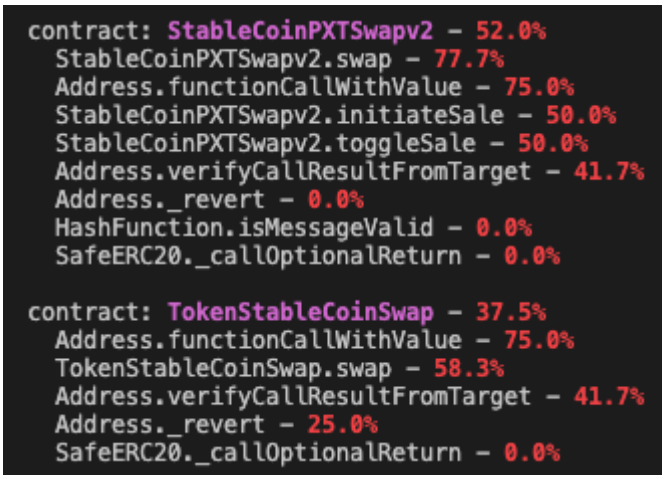
Testing coverage

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

Modus contracts tests:







```
tests/test_kyc_module.py::test_add_potential_modus RUNNING
Transaction sent: 0xa608d3fbd5a6a7f323fbecfd6fe6adde8e1e55ca529f456f34202188219010ce
Transaction sent: 0x264eaf64d23ad747a5279b34ba34111bce4f8ba40dfc05b25cc86552da9a29be
tests/test_kyc_module.py::test_add_potential_modus PASSED
tests/test_modus_registry.py::test_add_potential_modus RUNNING
Transaction sent: 0xe50405f04ceb89772330c26724c80b9013a75ab12b23f05ee8140ebe36545b33
Transaction sent: 0x972ab43f1cce1e1f568f4a2ffe213be0fdcc8796f291ea652536dadca108397b
Transaction sent: 0x470f3b403d30359f55b91fae9ef59da6016152b41dbb4ec7d7e20256107d3503
tests/test_modus_registry.py::test_add_potential_modus PASSED
tests/test_modus_registry.py::test_accept_modus_admin_nomination RUNNING
Transaction sent: 0x3e3c5ef1a39246018df66c7ac929d9b2f4b34568313a1f85fa8c9b2f65c09b02
tests/test_modus_registry.py::test_accept_modus_admin_nomination PASSED
tests/test_modus_registry.py::test_set_kyc_admin RUNNING
Transaction sent: 0x0ae912423b9f8cd6df7be4d07af429c3348142c8891a06fd069f23b32a019583
Transaction sent: 0x68aea2756e3492f9e1fb53a6a2dd53a4ef66450e8f557e52fa0c7d83e8690fea
tests/test_modus_registry.py::test_set_kyc_admin PASSED
tests/test_modus_registry.py::test_set_proxy_admin RUNNING
Transaction sent: 0xfbe6c56d143e12fd8ade0d5c91ab374934abb79251ba9beceb3244e857c2cb3a
Transaction sent: 0xb699cecf5b323475cd3e2a710a8c586e414904753e6e958b00709be33de7dc11
tests/test_modus_registry.py::test_set_proxy_admin PASSED
tests/test_modus_registry.py::test_set_modus_factory RUNNING
Transaction sent: 0xb71bba14b5fbc12dcf469ea950e84317f59cff1077ab3538ebfe2c6ef5a5a0b9
Transaction sent: 0x016ad0b74fc4949418c768527b6e5cfb6056cd7240fe34f5243f5adcb42c1cea
tests/test_modus_registry.py::test_set_modus_factory PASSED
tests/test_modus_registry.py::test_set_schema RUNNING
Transaction sent: 0x74d9d5ab1c47109f5baa43a83421d88d900864f3fcf07ede61207fa44c332302
Transaction sent: 0x9f29945b28b5ac9c8a6278de8ab37bb71df463b1d1e5ec858b94f0bd6ca76203
tests/test_modus_registry.py::test_set_schema PASSED
```

```
tests/test_modus_registry.py::test_set_project_details RUNNING
Transaction sent: 0x859928e538c328de5772567e2e2b8423b076139b3ca926d95e34bd5961a1fc04
Transaction sent: 0x9c0da8eba340f7a10e0e586c174f68b6fa575bee6bb699be893c52e1158fdf81
Transaction sent: 0x4b042e09860a4b5a50a1628517ba1ef8a176b98e4718ccbc033e9a34f3f31d55
tests/test_modus_registry.py::test_set_project_details PASSED
tests/test_modus_registry.py::test_blacklist_account RUNNING
Transaction sent: 0x70ec6489466bdffb82e3053ea45cab7b41ef1c5dd5ef883498a494ada0f87d3c
tests/test_modus_registry.py::test_blacklist_account PASSED
tests/test_pxt_staking.py::test_stake RUNNING
Transaction sent: 0x5cd3ec35ee4f135b592f5dd12ef8d93a28e6013c3936ae1beeaca14783ff46e3
Transaction sent: 0xd279eb8cde081c2500409780d3b7387b0df6cb29c4ca1e82967b3b99ac4ca413
Transaction sent: 0x05b9132f2b00b9d653b72d4b41302a39170a0e75d42061c20f40bdd989f0da80
Transaction sent: 0x6db9b7336d31a3e2cc2e563d6b244fcb6dc9ee515b6ca60d343a871c2d8ca1f4
Transaction sent: 0x182b525af3d68806841aa1f1125e0d4ee47a3d7cfd61f5220f3bd0f96b9a0cd4
tests/test_pxt_staking.py::test_stake PASSED
tests/test_pxt_staking.py::test_unstake RUNNING
Transaction sent: 0x2669d223832db6ed22f9f6ad0ddf266e9469ee2fa1d92a50e52b5be576e6dbb0
Transaction sent: 0x17c03f1bfe284743dc5a72884b31f7864d8be1e61b0c7d26beceed3b8c875f4e
Transaction sent: 0x9cb594391f213ec229da3106aebfd848104f8592f505105425246f7e0651536e
Transaction sent: 0xae6487c9061747ea5f4dd72198f9254cd0fddad82efabb25267bb54304b95810
tests/test_pxt_staking.py::test_unstake PASSED
tests/test_pxt_staking.py::test_collect_remaining_return_tokens RUNNING
Transaction sent: 0xf77a251fb0934a03c8fed3ca0f136bc1e32d6e49ea7eafcff74e1725ab7ee46f
Transaction sent: 0xbf36aab1ad94546b0527494d1305894bd63dd28ddd597c45dec180c3c1466f8f
tests/test_pxt_staking.py::test_collect_remaining_return_tokens PASSED
tests/test_pxt_staking.py::test_collect_remaining_modus_from_swap RUNNING
Transaction sent: 0x5035bd5c3b8d246cc3e233f0b218c67ad4b9414d152ef1d33e4797583de5ee7c
Transaction sent: 0x2d0f25dee2112e7a8089b336401116936888424a8026c37fd43290761b24dfc2
tests/test_pxt_staking.py::test_collect_remaining_modus_from_swap PASSED
tests/test_pxt_staking.py::test_admin_unstake_for_lost_wallet RUNNING
Transaction sent: 0x3d57a2ea391487483ae2f3e328e24cf35ef9c3082633aa12494690af237c51dc
Transaction sent: 0xd8b1ab51f6cc33e9e27d7f5481d07a202faa4eb93f6cd773abfeeeed6a417e3c
Transaction sent: 0x2497a90ea6553715a523a49d8d53eb48be1410b61aa43893889095f2d8b72f3d
Transaction sent: 0x264ad4461cc6e623eb9e66c27583317f7533c49db2432032ecd64e179eb403f5
Transaction sent: 0x60f6e170d525d5d4bd8484f7c3f3fbe03f50fd09e2b37baf54b009a591cacd1d
tests/test_pxt_staking.py::test_admin_unstake_for_lost_wallet PASSED
```

```
tests/test_stable_coin_pxt.py::test_initiate_sale RUNNING
Transaction sent: 0x65ebc00d09d9cf554b8dec8325c5ec885fceb0f8ab73a186f85244a0a12e6336
Transaction sent: 0xcb4a118689c3126e1a2014dea522ae1b24eea0e2be752ba921e4c4d412aadd15
Transaction sent: 0x407308e6b7fe546e7269b539067b22cedaf24251c7c709a38c6a259dd52320de
tests/test_stable_coin_pxt.py::test_initiate_sale PASSED
tests/test_stable_coin_pxt.py::test_swap RUNNING
Transaction sent: 0x4e052159f0cfa84230c21b7cf99b3a683c8dc78ae681a0c9fef0cc678f16f678
Transaction sent: 0x49582f3945d7c66b2064aac4f09de4f051e3ecfeb8976f7243f132367b1f791c
Transaction sent: 0x09d854264989b5dae31545d18c93bf223775bf34414cf418e48a998b6e77142a
Transaction sent: 0x608de175c7482d4c5e7b576b9a4aa653833602d638930728087cf7f160ca0ee7
Transaction sent: 0xa00a28ae995a58f952e9732cd237942023a3fa62092e41f72bad764e1274c9ec
Transaction sent: 0xa7b9345d6c4e9ea493951be4147ede6768990ac991d990391b56dac64bfaa03a
tests/test_stable_coin_pxt.py::test_swap PASSED

========================================================================================================== 16 passed
```

Annexes

Testing code:

test_kyc_module:

```
from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
)

from scripts.deploy import (
    deploy_modus_registry,
    deploy_kyc
)

def test_add_potential_modus(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    kyc = deploy_kyc(owner, mr.address)

    # assert
    with reverts("KYC: Access Denied"):
        kyc.setKYCStatus(
            ["0x2B7E1A2C726dc3271066362561872bd512AfAcdE",
"0xd1862382094485Ee8E953218f4e644fc502434CA"],
            [True, True],
            {"from": owner})
    with reverts("Index out of range"):
        kyc.setKYCStatus(
            ["0x2B7E1A2C726dc3271066362561872bd512AfAcdE",
"0xd1862382094485Ee8E953218f4e644fc502434CA"],
            [True],
            {"from": admin})

    assert kyc.getKYCStatus("0x2B7E1A2C726dc3271066362561872bd512AfAcdE") == False
```

```
    kyc.setKYCStatus(
        ["0x2B7E1A2C726dc3271066362561872bd512AfAcdE",
"0xd1862382094485Ee8E953218f4e644fc502434CA"],
        [True, True],
        {"from": admin})
    assert kyc.getKYCStatus("0x2B7E1A2C726dc3271066362561872bd512AfAcdE") == True

    mr.setKYCAdmin(other, {"from": admin})
    assert kyc.getKYCStatus("0xEC1cA6610C7bC4f8C9af198E067683Aae76A8C39") == False
    kyc.setKYCStatus(
        ["0xEC1cA6610C7bC4f8C9af198E067683Aae76A8C39"],
        [True],
        {"from": other})
    assert kyc.getKYCStatus("0xEC1cA6610C7bC4f8C9af198E067683Aae76A8C39") == True
```

test_modus_registry:

```
from brownie import (
    reverts,
)

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

from scripts.deploy import (
    deploy_modus_registry
)

def test_add_potential_modus(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
```

```
    # assert
    with reverts("ModusRegistry: Not ModusAdmin"):
        mr.addPotentialModusAdmin(other, {"from": owner})
    with reverts("ModusRegistry: Zero Address"):
        mr.addPotentialModusAdmin(ZERO_ADDRESS, {"from": admin})
    with reverts("ModusRegistry: Potential Admin should not be current Admin"):
        mr.addPotentialModusAdmin(admin, {"from": admin})
    assert mr.potentialModusAdmin() == ZERO_ADDRESS
    mr.addPotentialModusAdmin(other, {"from": admin})
    assert mr.potentialModusAdmin() == other

def test_accept_modus_admin_nomination(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    # assert
    with reverts("ModusRegistry: Only the potential admin can accept nomination"):
        mr.acceptModusAdminNomination({"from": owner})
    mr.addPotentialModusAdmin(other, {"from": admin})
    assert mr.potentialModusAdmin() == other
    mr.acceptModusAdminNomination({"from": other})
    assert mr.modusAdmin() == other
    assert mr.potentialModusAdmin() == ZERO_ADDRESS

def test_set_kyc_admin(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    # assert
    with reverts("ModusRegistry: Not ModusAdmin"):
        mr.setKYCAdmin(other, {"from": owner})
    with reverts("ModusRegistry: Zero Address"):
        mr.setKYCAdmin(ZERO_ADDRESS, {"from": admin})
    assert mr.kycAdmin() == ZERO_ADDRESS
    mr.setKYCAdmin(other, {"from": admin})
    assert mr.kycAdmin() == other
```

```
def test_set_proxy_admin(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    # assert
    with reverts("ModusRegistry: Not ModusAdmin"):
        mr.setProxyAdmin(other, {"from": owner})
    with reverts("ModusRegistry: Zero Address"):
        mr.setProxyAdmin(ZERO_ADDRESS, {"from": admin})
    assert mr.proxyAdmin() == ZERO_ADDRESS
    mr.setProxyAdmin(other, {"from": admin})
    assert mr.proxyAdmin() == other

def test_set_modus_factory(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    # assert
    with reverts("ModusRegistry: Not ModusAdmin"):
        mr.setModusFactory(other, {"from": owner})
    with reverts("ModusRegistry: Zero Address"):
        mr.setModusFactory(ZERO_ADDRESS, {"from": admin})
    assert mr.modusFactory() == ZERO_ADDRESS
    mr.setModusFactory(other, {"from": admin})
    assert mr.modusFactory() == other

def test_set_schema(only_local):
    #arrange
    owner = get_account(0)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    max_days = 10
```

```
    ex_base = 2
    percentatge_reward = 1000
    # assert
    with reverts("ModusRegistry: Not ModusAdmin"):
        mr.setScheme(max_days, ex_base, percentatge_reward, {"from": owner})
    with reverts("ModusRegistry: Reward Percentage cannot be more than 100%"):
        mr.setScheme(max_days, ex_base, 10001, {"from": admin})
    mr.setScheme(max_days, ex_base, percentatge_reward, {"from": admin})

def test_set_project_details(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    # assert
    mr.setModusFactory(other, {"from": admin})
    project_id = 1
    stage_three_data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3"
    stage_four_data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3"

    with reverts("ModusRegistry: Not ModusFactory"):
        mr.setProjectDetails(project_id, 2, b'random_data', {"from": owner})
    with reverts("ModusRegistry: Function Mismatch"):
        mr.setProjectDetails(project_id, 1, b'random_data', {"from": other})

    mr.setProjectDetails(project_id, 3, stage_three_data, {"from": other})
    mr.setProjectDetails(project_id, 4, stage_four_data, {"from": other})

    with reverts("ModusRegistry: Function Mismatch"):
        mr.setProjectDetails(project_id, 2, b'random_data', b'random_data', {"from": other})

    data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000001"
    mr.setProjectDetails(project_id, 1, data, stage_three_data, {"from": other})
```

```
def test_blacklist_account(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)

    # assert
    mr.setModusFactory(other, {"from": admin})
    project_id = 1
    stage_three_data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3"

    with reverts("ModusRegistry: Caller is not valid PXTStaking"):
        mr.blackListAccount(project_id, lost_wallet_admin, {"from": other})

    data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000001"
    mr.setProjectDetails(project_id, 1, data, stage_three_data, {"from": other})

    assert mr.isAccountBlackListed(lost_wallet_admin) == False
    mr.blackListAccount(project_id, lost_wallet_admin, {"from":
"0x21b42413bA931038f35e7A5224FaDb065d297Ba3"})
    assert mr.isAccountBlackListed(lost_wallet_admin) == True
```

test_pxt_staking:

```
from eth_abi import encode

from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    DAY_TIMESTAMP,
    get_account
)

from scripts.deploy import (
```

```
    deploy_modus_registry,
    deploy_pxt_staking,
    deploy_stable_coin_pxt,
    deploy_pxt, deploy_pxrt,
    deploy_stable_coin,
    deploy_token_stable_coin_swap,
    deploy_kyc
)

def test_stake(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    investor = get_account(5)
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    pxt = deploy_pxt_staking(owner)
    pxt.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
    fake_stable_coin_addr = "0x7FF5A7A91ce1eDB558955Ed8E594FDbd8ad13404"
    fake_pxt_addr = "0xb1f57b8529F3F6EFB8eBd93db9edb205C2c04EE5"
    # assert
    with reverts("ProxyUtility: Not ModusFactory"):
        pxt.cancelProject({"from": admin})
    mr.setModusFactory(other, {"from": admin})
    pxt.cancelProject({"from": other})
    with reverts("PXTStaking: Project cancelled"):
        pxt.stake(investor, 1e17, {"from": admin})

    pxt = deploy_pxt_staking(owner)
    pxt.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
    mr.setModusFactory(other, {"from": admin})
    pxt.endProject({"from": other})
    with reverts("PXTStaking: Project is not active"):
        pxt.stake(investor, 1e17, {"from": admin})

    pxt = deploy_pxt_staking(owner)
    pxt.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
    token_data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000007FF5A7A91ce1eDB558955Ed8E594FDbd8ad13404000000000000000000000000000000b1f57b8529F3F6EFB8eBd93db9edb205C2c04EE5"
    data =
"0x000000000000000000000000021b42413ba931038f35e7a5224fadb065d297ba3000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
```

```
00000000000000000000001"
    mr.setProjectDetails(1, 1, data, token_data, {"from": other})
    mr.blackListAccount(1, lost_wallet_admin, {"from": fake_pxt_addr})
    with reverts("PXTStaking: Account is Blacklisted"):
        pxt.stake(lost_wallet_admin, 1e17, {"from": admin})
    with reverts("PXTStaking: Caller not StableCoinPXTSwap"):
        pxt.stake(investor, 1e17, {"from": admin})
    tx = pxt.stake(investor, 1e17, {"from": fake_stable_coin_addr})
    assert tx.events["PXTStaked"] is not None
    assert tx.events["PXTStaked"][0]['projectID'] == 1
    assert tx.events["PXTStaked"][0]['investor'] == investor
    assert tx.events["PXTStaked"][0]['amount'] == 1e17
    assert pxt.pxtBalance() == 1e17
    tx = pxt.stake(investor, 1e17, {"from": fake_stable_coin_addr})
    assert tx.events["PXTStaked"] is not None
    assert tx.events["PXTStaked"][0]['projectID'] == 1
    assert tx.events["PXTStaked"][0]['investor'] == investor
    assert tx.events["PXTStaked"][0]['amount'] == 1e17
    assert pxt.pxtBalance() == 2e17

def test_unstake(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    investor = get_account(5)
    custodian_wallet = get_account(6)
    operational_wallet = get_account(7)

    # Modus Registry
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    # Stable coin
    stable_coin = deploy_stable_coin(owner)
    # STABLE COIN PXT SWAP
    stable_coin_swap = deploy_stable_coin_pxt(owner)
    pxt_stable_coin = deploy_stable_coin_pxt(owner)
    pxrt_stable_coin = deploy_stable_coin_pxt(owner)
    # PXT
    pxt = deploy_pxt(owner)
    pxt.initialize(1, mr.address, 9)
    # PXT Staking
    pxt_staking = deploy_pxt_staking(owner)
    pxt_staking.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
    # TOKEN SWAP 1
```

```
    pxt_token_swap = deploy_token_stable_coin_swap(owner)
    pxt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)
    # TOKEN SWAP 2
    pxrt_token_swap = deploy_token_stable_coin_swap(owner)
    pxrt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)

    # assert
    mr.setModusFactory(other, {"from": admin})
    # stage 1
    data = encode(['address', 'uint256', 'uint256'], [str(custodian_wallet), 0, 1])
    token_data = encode(
        ['address', 'address', 'address'],
        [pxt.address, stable_coin_swap.address, pxt_staking.address])
    mr.setProjectDetails(1, 1, data, token_data, {"from": other})
    # PXRT
    pxrt = deploy_pxrt(owner)
    pxrt.initialize(1, 100e18, 10e18, operational_wallet, mr.address, 18)
    # stage 3 & 4
    data = encode(
        ['address', 'address', 'address'],
        [pxrt.address, pxt_token_swap.address, pxrt_token_swap.address])
    mr.setProjectDetails(1, 3, data, {"from": other})
    data = encode(
        ['address', 'address'],
        [pxt_stable_coin.address, pxrt_stable_coin.address])
    mr.setProjectDetails(1, 4, data, {"from": other})

    with reverts("PXTStaking: Insuffecient funds"):
        pxt_staking.unstake(1e17, {"from": investor})

    pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})
    pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})

    with reverts():
        pxt_staking.unstake(1e17, {"from": investor})

    pxt_staking.endProject({"from": other})
    with reverts():
        pxt_staking.unstake(1e17, {"from": investor})

    pxt_staking.cancelProject({"from": other})
    with reverts():
        pxt_staking.unstake(1e17, {"from": investor})

def test_collect_remaining_return_tokens(only_local):
    #arrange
```

```
owner = get_account(0)
other = get_account(1)
admin = get_account(2)
wallet = get_account(3)
lost_wallet_admin = get_account(4)
investor = get_account(5)
custodian_wallet = get_account(6)
operational_wallet = get_account(7)

# Modus Registry
mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
# Stable coin
stable_coin = deploy_stable_coin(owner)
# STABLE COIN PXT SWAP
stable_coin_swap = deploy_stable_coin_pxt(owner)
pxt_stable_coin = deploy_stable_coin_pxt(owner)
pxrt_stable_coin = deploy_stable_coin_pxt(owner)
# PXT
pxt = deploy_pxt(owner)
pxt.initialize(1, mr.address, 9)
# PXT Staking
pxt_staking = deploy_pxt_staking(owner)
pxt_staking.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
# TOKEN SWAP 1
pxt_token_swap = deploy_token_stable_coin_swap(owner)
pxt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)
# TOKEN SWAP 2
pxrt_token_swap = deploy_token_stable_coin_swap(owner)
pxrt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)

# assert
mr.setModusFactory(other, {"from": admin})
# stage 1
data = encode(['address', 'uint256', 'uint256'], [str(custodian_wallet), 0, 1])
token_data = encode(
    ['address', 'address', 'address'],
    [pxt.address, stable_coin_swap.address, pxt_staking.address])
mr.setProjectDetails(1, 1, data, token_data, {"from": other})
# PXRT
pxrt = deploy_pxrt(owner)
pxrt.initialize(1, 100e18, 10e18, operational_wallet, mr.address, 18)
# stage 3 & 4
data = encode(
    ['address', 'address', 'address'],
    [pxrt.address, pxt_token_swap.address, pxrt_token_swap.address])
mr.setProjectDetails(1, 3, data, {"from": other})
```

```
    data = encode(
        ['address', 'address'],
        [pxt_stable_coin.address, pxrt_stable_coin.address])
    mr.setProjectDetails(1, 4, data, {"from": other})

    pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})
    pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})

    with reverts("PXTStaking: Caller is neither ModusAdmin or Modus Treasury"):
        pxt_staking.collectRemainingReturnTokens({"from": operational_wallet})

    with reverts():
        pxt_staking.collectRemainingReturnTokens({"from": admin})

def test_collect_remaining_modus_from_swap(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    investor = get_account(5)
    custodian_wallet = get_account(6)
    operational_wallet = get_account(7)

    # Modus Registry
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    # Stable coin
    stable_coin = deploy_stable_coin(owner)
    # STABLE COIN PXT SWAP
    stable_coin_swap = deploy_stable_coin_pxt(owner)
    pxt_stable_coin = deploy_stable_coin_pxt(owner)
    pxrt_stable_coin = deploy_stable_coin_pxt(owner)
    # PXT
    pxt = deploy_pxt(owner)
    pxt.initialize(1, mr.address, 9)
    # PXT Staking
    pxt_staking = deploy_pxt_staking(owner)
    pxt_staking.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
    # TOKEN SWAP 1
    pxt_token_swap = deploy_token_stable_coin_swap(owner)
    pxt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)
    # TOKEN SWAP 2
    pxrt_token_swap = deploy_token_stable_coin_swap(owner)
    pxrt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)
```

```
    # assert
    mr.setModusFactory(other, {"from": admin})
    # stage 1
    data = encode(['address', 'uint256', 'uint256'], [str(custodian_wallet), 0, 1])
    token_data = encode(
        ['address', 'address', 'address'],
        [pxt.address, stable_coin_swap.address, pxt_staking.address])
    mr.setProjectDetails(1, 1, data, token_data, {"from": other})
    # PXRT
    pxrt = deploy_pxrt(owner)
    pxrt.initialize(1, 100e18, 10e18, operational_wallet, mr.address, 18)
    # stage 3 & 4
    data = encode(
        ['address', 'address', 'address'],
        [pxrt.address, pxt_token_swap.address, pxrt_token_swap.address])
    mr.setProjectDetails(1, 3, data, {"from": other})
    data = encode(
        ['address', 'address'],
        [pxt_stable_coin.address, pxrt_stable_coin.address])
    mr.setProjectDetails(1, 4, data, {"from": other})

    pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})
    pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})

    with reverts("PXRTModusSwap: Caller is not Modus Treasury"):
        pxt_staking.collectRemainingModusfromSwapContracts({"from": operational_wallet})

    with reverts():
        pxt_staking.collectRemainingModusfromSwapContracts({"from": wallet})

def test_admin_unstake_for_lost_wallet(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    investor = get_account(5)
    custodian_wallet = get_account(6)
    operational_wallet = get_account(7)

    # Modus Registry
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    kyc = deploy_kyc(owner, mr.address)
    mr.setKycModule(kyc.address, {"from": admin})
    # Stable coin
```

```
stable_coin = deploy_stable_coin(owner)
# STABLE COIN PXT SWAP
stable_coin_swap = deploy_stable_coin_pxt(owner)
pxt_stable_coin = deploy_stable_coin_pxt(owner)
pxrt_stable_coin = deploy_stable_coin_pxt(owner)
# PXT
pxt = deploy_pxt(owner)
pxt.initialize(1, mr.address, 9)
# PXT Staking
pxt_staking = deploy_pxt_staking(owner)
pxt_staking.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
# TOKEN SWAP 1
pxt_token_swap = deploy_token_stable_coin_swap(owner)
pxt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)
# TOKEN SWAP 2
pxrt_token_swap = deploy_token_stable_coin_swap(owner)
pxrt_token_swap.initialize(mr.address, 1, 10, stable_coin.address)

# assert
mr.setModusFactory(other, {"from": admin})
# stage 1
data = encode(['address', 'uint256', 'uint256'], [str(custodian_wallet), 0, 1])
token_data = encode(
    ['address', 'address', 'address'],
    [pxt.address, stable_coin_swap.address, pxt_staking.address])
mr.setProjectDetails(1, 1, data, token_data, {"from": other})
# PXRT
pxrt = deploy_pxrt(owner)
pxrt.initialize(1, 100e18, 10e18, operational_wallet, mr.address, 18)
# stage 3 & 4
data = encode(
    ['address', 'address', 'address'],
    [pxrt.address, pxt_token_swap.address, pxrt_token_swap.address])
mr.setProjectDetails(1, 3, data, {"from": other})
data = encode(
    ['address', 'address'],
    [pxt_stable_coin.address, pxrt_stable_coin.address])
mr.setProjectDetails(1, 4, data, {"from": other})

with reverts("PXTStaking: Insuffecient funds"):
    pxt_staking.unstake(1e17, {"from": investor})

pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})
pxt_staking.stake(investor, 1e17, {"from": stable_coin_swap.address})

with reverts("ModusStaking: KYC not verified"):
```

```
        pxt_staking.adminUnstakeForLostwallet(investor, operational_wallet, {"from": admin})

    kyc.setKYCStatus(
            [operational_wallet],
            [True],
            {"from": admin})

    with reverts("PXTStaking: Not LostWallet admin"):
        pxt_staking.adminUnstakeForLostwallet(investor, operational_wallet, {"from": admin})
    with reverts():
        pxt_staking.adminUnstakeForLostwallet(investor, operational_wallet, {"from": lost_wallet_admin})
    pxt_staking.cancelProject({"from": other})
    with reverts():
        pxt_staking.adminUnstakeForLostwallet(investor, operational_wallet, {"from": lost_wallet_admin})
```

test_stable_coin_pxt:

```
from eth_abi import encode

from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    DAY_TIMESTAMP,
    get_account
)

from scripts.deploy import (
    deploy_modus_registry,
    deploy_pxt_staking,
    deploy_stable_coin_pxt,
    deploy_pxt, deploy_pxrt,
    deploy_stable_coin,
    deploy_token_stable_coin_swap,
    deploy_kyc
)

def test_initiate_sale(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
```

```
    fake_proof_verifier = get_account(5)

    # Modus Registry
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    mr.setModusFactory(other, {"from": admin})

    stable_coin_pxt = deploy_stable_coin_pxt(owner)
    stable_coin_pxt.initialize(
        1, 10e18, 10, mr.address, fake_proof_verifier,
        {"from": owner}
    )

    # assert
    with reverts("ProxyUtility: Not ModusFactory"):
        stable_coin_pxt.initiateSale({"from": admin})
    stable_coin_pxt.initiateSale({"from": other})
    with reverts("StableCoinPXTSwap: Already Initiated"):
        stable_coin_pxt.initiateSale({"from": other})
    stable_coin_pxt.toggleSale({"from": other})
    with reverts("ProxyUtility: Neither ModusAdmin or ModusFactory"):
        stable_coin_pxt.toggleSale({"from": wallet})

def test_swap(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    admin = get_account(2)
    wallet = get_account(3)
    lost_wallet_admin = get_account(4)
    fake_proof_verifier = get_account(5)
    extra = get_account(6)
    custodian_wallet = get_account(7)

    # Modus Registry
    mr = deploy_modus_registry(owner, wallet, admin, lost_wallet_admin)
    mr.setModusFactory(other, {"from": admin})

    stable_coin_pxt = deploy_stable_coin_pxt(owner)
    stable_coin_pxt.initialize(
        1, 10e18, 10, mr.address, fake_proof_verifier,
        {"from": owner}
    )

    # PXT Staking
    pxt_staking = deploy_pxt_staking(owner)
    pxt_staking.initialize(DAY_TIMESTAMP, 1e18, 1, mr.address, {"from": owner})
```

```
# PXT
pxt = deploy_pxt(owner)
pxt.initialize(1, mr.address, 9)

data = encode(['address', 'uint256', 'uint256'], [str(custodian_wallet), 0, 1])
token_data = encode(
    ['address', 'address', 'address'],
    [pxt.address, stable_coin_pxt.address, pxt_staking.address])
mr.setProjectDetails(1, 1, data, token_data, {"from": other})

# assert
random_sig = b'random_signature'
mr.blackListAccount(1, extra, {"from": pxt_staking.address})
with reverts("StableCoinPXTSwap: Address Blacklisted"):
    stable_coin_pxt.swap(1e18, random_sig, {"from": extra})
mr.whiteListAccount(extra, {"from": admin})
with reverts("StableCoinPXTSwap: Sale yet to Begin"):
    stable_coin_pxt.swap(1e18, random_sig, {"from": extra})
stable_coin_pxt.initiateSale({"from": other})

with reverts("StableCoinPXTSwap: Investment exceeds the required Capital"):
    stable_coin_pxt.swap(15e18, random_sig, {"from": extra})

with reverts("StableCoinPXTSwap: Not a Premium Investor"):
    stable_coin_pxt.swap(1e18, random_sig, {"from": extra})

stable_coin_pxt.setPremiumInvestor(extra, True, 1e18/2, {"from": admin})
with reverts("StableCoinPXTSwap: Exceeds Maximum stake amount"):
    stable_coin_pxt.swap(1e18, random_sig, {"from": extra})
stable_coin_pxt.setPremiumInvestor(extra, True, 1e18, {"from": admin})
with reverts():
    stable_coin_pxt.swap(1e18, random_sig, {"from": extra})
```

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	1	1				
Medium	2	2				
Low	4	4				
Informational	0	0				

Assessment Results

Score Results

Review	Score
Audit Score	**65/100**
Assure KYC	Pending
Audit Score	70/100

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 Fail

Following our comprehensive security audit of the project Modus, we regret to inform you that the project has failed to meet the required security standards. The audit has uncovered three high+medium vulnerabilities that pose significant risks.
These vulnerabilities not only compromise the security of the platform but also threaten the integrity and trustworthiness of the system. It is imperative to address these issues immediately before deploying the contracts.
We are prepared to assist in the remediation of these vulnerabilities to enhance the security posture of Modus. Please contact us to discuss the next steps in detail.

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.