Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

GemPadLock

Date: 21/12/2024

Audit Status: FAIL

Audit Edition: Advanced

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

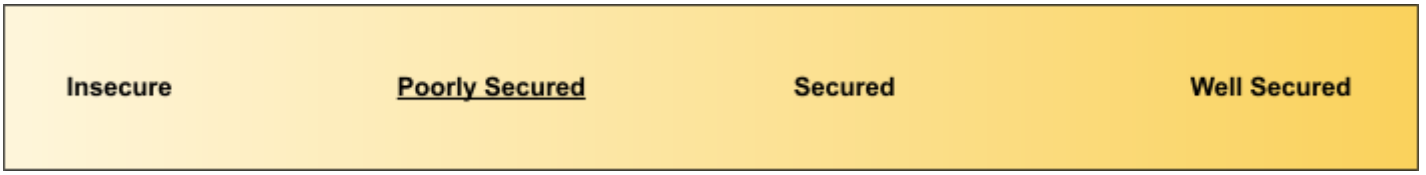
Risk Analysis

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Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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



Scope

Target Code And Revision

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

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	https://bscscan.com/address/0x06900552c7aDB5ac198eBb2641057337FD9996bB#code
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. Unrestricted Fee Setting in updateFee() Could Lead to Excessive Charges

Issue: Within the updateFee() function, there is no cap on the fee parameters—projectCreationFee, lpTokenNormalLockFee, lpTokenVestingLockFee, normalTokenNormalLockFee, and normalTokenVestingLockFee. This means the contract owner can set arbitrarily high fees, potentially resulting in unfair or exploitative costs for users.

Recommendation: Enforce a maximum allowable fee by adding a require() statement. This will ensure that any fee adjustments remain within a reasonable range and protect users from excessively high costs.

2. Potential Out-of-Gas with Large Inputs in multipleLock() and multipleVestingLock()

Issue: The functions multipleLock() and multipleVestingLock() allow creating multiple locks within a single transaction. If the arrays owners and amounts are excessively large, the loop processing these arrays may consume too much gas, leading to a potential out-of-gas error.

Recommendation: Enforce a maximum array length for both owners and amounts to ensure that the transaction remains within gas limits. By imposing such a restriction, you prevent scenarios where excessively large inputs could cause the contract to run out of gas during execution.

3. Insecure UUPS Upgrade Mechanism

Issue: The contract inherits from UUPSUpgradeable but does not implement an _authorizeUpgrade() function. As a result, the standard UUPS “guard” is missing and may allow any external actor to call upgradeTo(...) or upgradeToAndCall(...)—depending on how the parent contract is structured. In many UUPS patterns, _authorizeUpgrade() must explicitly restrict upgrades to only the contract owner or a trusted role.

Recommendation: Implement a properly secured _authorizeUpgrade(newImplementation) that includes an authorization check (e.g., require(owner() == _msgSender(), "Not authorized");). Also ensure any intended upgrade path is correctly integrated (such as calling __UUPSUpgradeable_init() if needed). If the contract does not need to be upgradeable at all, consider removing or disabling UUPS references to eliminate any exploit paths.

1. Missing Zero-Address Check in updateAvailabilityForNFT()

Issue: In the function updateAvailabilityForNFT(), there is no validation to ensure that the NFT address passed (nft) is not the zero address. Consequently, the contract owner can mistakenly (or maliciously) set the availability of a zero address, which is generally an invalid address for NFTs.

Recommendation: Add a require() statement to verify that the nft address is not zero.

2. Excess ETH Fee Not Refunded

Issue: When a user pays fees via _payFee(), there is a require(msg.value >= _fee, ...), but any excess ETH is not refunded. This could result in users accidentally overpaying fees and losing the difference if they send more ETH than required.

Recommendation: Consider adding logic to refund msg.sender any msg.value in excess of the required fee. For example, storing uint256 overpaid = msg.value - _fee and then returning that amount to the sender.

3. Lack of Fallback/Receive Function

Issue: The contract does not define a fallback or receive function. In case a user or contract sends ETH accidentally (above the fee amount) or calls an unrecognized function, that ETH may become permanently stuck.

Recommendation: Although not critical, you might add a fallback or receive() function to handle unexpected ETH transfers. Alternatively, consider implementing a withdraw/claim function accessible by the owner (or a designated account) to rescue stuck ETH.

4. Project Ownership Auto-Assignment

Issue: When someone locks a project token for the first time and projects[projectToken].owner == address(0), the contract sets projects[projectToken].owner = msg.sender. This might cause confusion if an unauthorized user is the first to lock tokens for that project. While not a vulnerability per se, it can lead to misunderstandings about who “owns” the project in this contract's context

Recommendation: Communicate clearly in documentation that the first caller to lock a token in this contract becomes the “project owner” for that token in the GempadLock system, or restrict the initial project owner assignment to ensure only the legitimate project owner can perform the first lock.

5. Upgradeable Path Not Implemented

Issue: The contract inherits UUPSUpgradeable references and uses OwnableUpgradeable, but _authorizeUpgrade() is not clearly present. If future upgrades are intended, you must ensure _authorizeUpgrade(newImplementation) is properly implemented. Otherwise, the upgrade mechanism may be non-functional or insecure.

Recommendation: If the intention is truly to be upgradeable, add and restrict _authorizeUpgrade(newImplementation) to only the contract owner or a designated upgrader. If upgradeability is not required, consider removing UUPS references to avoid confusion.

6. Locking Deflationary Tokens Will Revert

Issue: The helper function _safeTransferFromEnsureExactAmount(...) requires that the recipient's balance increases by exact amount. Many deflationary, reflection, or fee-on-transfer tokens will deduct a fee in-transit, causing the recipient's balance to increase by a smaller amount. This causes the lock to revert.

Recommendation: If supporting deflationary tokens is desired, remove or adjust the exact balance check. If it is an intentional design choice to not support deflationary tokens, document this limitation clearly.

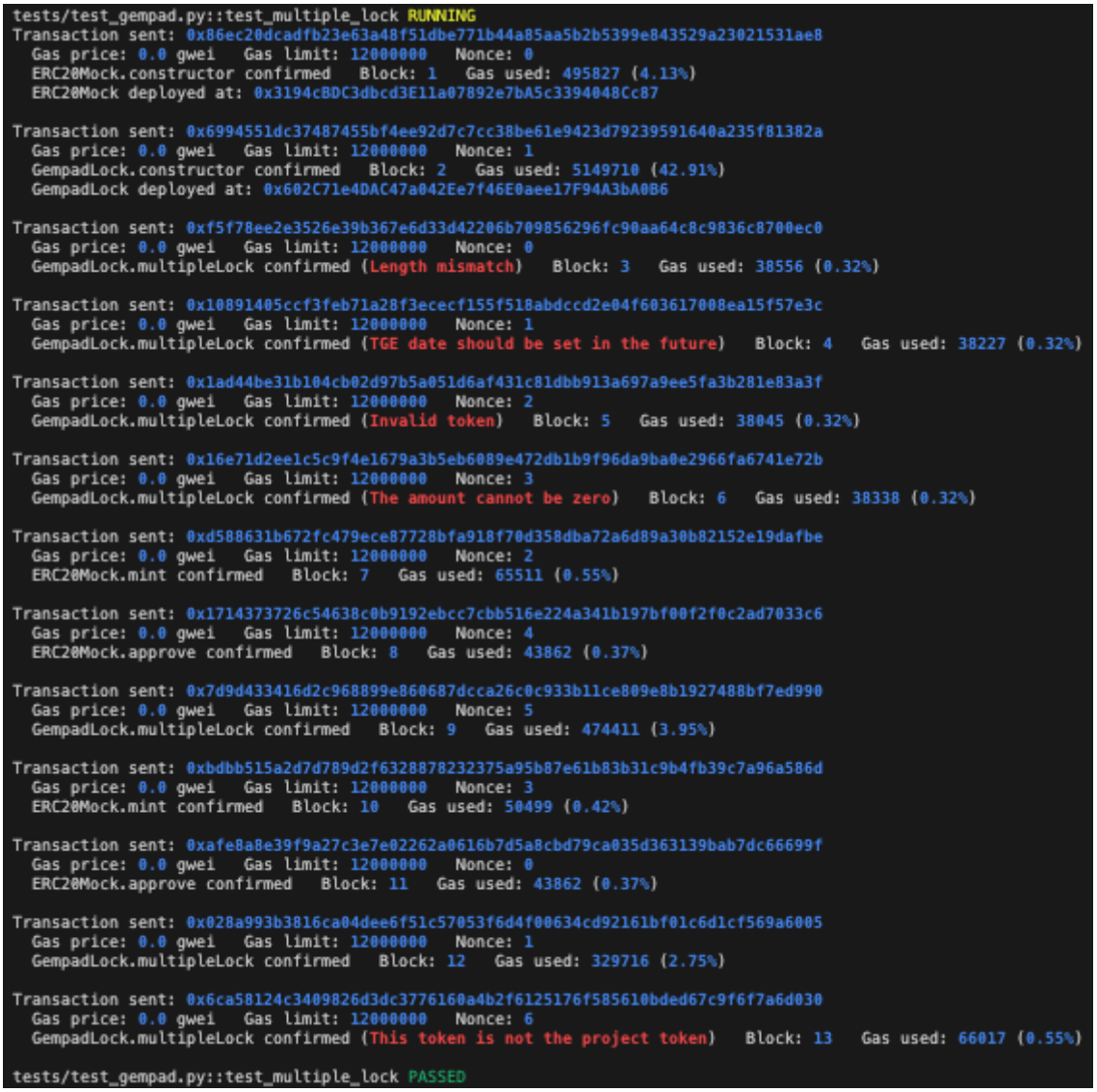
INFORMATIONAL

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.*

GemPadLock contract tests:



```
tests/test_gempad.py::test_multiple_vesting_lock RUNNING
Transaction sent: 0xdb6d7a788c1f8b4e4df70d6dd03844325ab85d47c6c41eb65ea7185b0cd3978a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  ERC20Mock.constructor confirmed   Block: 14   Gas used: 495827 (4.13%)
  ERC20Mock deployed at: 0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9

Transaction sent: 0xcf4deb56c6ef3530f5bc99be53670bf65f5557392366076d1801922f1c36de1a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  GempadLock.constructor confirmed   Block: 15   Gas used: 5149710 (42.91%)
  GempadLock deployed at: 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3

Transaction sent: 0x046b502271e0ce1665433a94a1bd7cbbfad410e27dc1e7e1c093c178019803f6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  GempadLock.multipleVestingLock confirmed (Invalid cycle)   Block: 16   Gas used: 38717 (0.32%)

Transaction sent: 0x70c645afbdb223adcd04e37b5f4ccbfee649810113e55c980e056a78ff045bdb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  GempadLock.multipleVestingLock confirmed (Invalid bips for TGE)   Block: 17   Gas used: 38753 (0.32%)

Transaction sent: 0xdc71d4ba4dba71a448f344e47c73cb99be6f01a14ecf16d26fd51a6c68217513
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  GempadLock.multipleVestingLock confirmed (Invalid bips for cycle)   Block: 18   Gas used: 38830 (0.32%)

Transaction sent: 0x89caac1b731c58973c8c010d45094cc5971e6049a1e1dfcc5a00d84430e045f8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  GempadLock.multipleVestingLock confirmed (Sum of TGE bps and cycle should be less than 10000)   Block: 19   Gas used: 38989 (0.32%)

Transaction sent: 0x44c9bb2e78c994ec7f319d349028f68ba0a75429d1a80284dc944d514e535fcd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  ERC20Mock.mint confirmed   Block: 20   Gas used: 65511 (0.55%)

Transaction sent: 0xb78f4e44c112dc342a967ba00900348e57e270f0ef0994004c7a296d4a57e5d9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  ERC20Mock.approve confirmed   Block: 21   Gas used: 43862 (0.37%)

Transaction sent: 0xbc74f591869569ebfa5644b697c6706b069024acf678a3db29bb305db48f0205
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  GempadLock.multipleVestingLock confirmed   Block: 22   Gas used: 475506 (3.96%)

Transaction sent: 0xdebe827a20320338cf20199ce549bbe7c339ea49c655b8476ab07d617594b2c8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  ERC20Mock.mint confirmed   Block: 23   Gas used: 50499 (0.42%)

Transaction sent: 0x57f793245c8ee442fbb1f9f3751dee654e2838d0c6b6826ca42cafe5acbdf512
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  ERC20Mock.approve confirmed   Block: 24   Gas used: 43862 (0.37%)

Transaction sent: 0x029695c8e49ee033ea28460273a2142ca4914f42c0ac7cd24dfdc044b4fb11cc
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  GempadLock.multipleVestingLock confirmed   Block: 25   Gas used: 315823 (2.63%)

tests/test_gempad.py::test_multiple_vesting_lock PASSED
```

```
tests/test_gempad.py::test_unlock RUNNING
Transaction sent: 0xfa03ce8f2c2a1b1ed42ce160b3eff94028c4f13d5a61b895a2bc2e41ee0aa220
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  ERC20Mock.constructor confirmed   Block: 26   Gas used: 495827 (4.13%)
  ERC20Mock deployed at: 0x420b1099B9eF5baba6D92029594eF45E19A04A4A

Transaction sent: 0x0fed4c834daddacccd3c8e5e46d1ac805bd0c89d3a7aced796ce47e4674d312f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  GempadLock.constructor confirmed   Block: 27   Gas used: 5149710 (42.91%)
  GempadLock deployed at: 0xa3B53dDCd2E3fC28e8E130288F2aBD8d5EE37472

Transaction sent: 0x054480572842f3a7c65927681252c40649180cd4f74ea1001fcb477fd52dba1b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  ERC20Mock.mint confirmed   Block: 28   Gas used: 65511 (0.55%)

Transaction sent: 0x639df01725a7fc832fa9e9be66b837f01781ee5891fdf685858be858364a3855
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  ERC20Mock.approve confirmed   Block: 29   Gas used: 43862 (0.37%)

Transaction sent: 0xcec3be04ade9d5c1d06df3d1e39a40144cd4fe258ded017b668c0380b659aecd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  GempadLock.multipleLock confirmed   Block: 30   Gas used: 474411 (3.95%)

Transaction sent: 0xfcc43b99a27eafc026a3ea0e8d4c587d7f30ab059020cceb9e908e78035bad8a
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  GempadLock.unlock confirmed (Invalid lock ID)   Block: 31   Gas used: 22618 (0.19%)

Transaction sent: 0x309c49710b11f0f2f6b59933047d1276744216d0d31e8b95d5eaa65947801176
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  GempadLock.unlock confirmed (You are not the owner of this lock)   Block: 32   Gas used: 30273 (0.25%)

Transaction sent: 0x05d322ded74e693d28e5090417c7792f2ba8d55eeb4942b0d11b8b659a41c735
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  GempadLock.unlock confirmed (The lock is not unlocked yet)   Block: 33   Gas used: 32060 (0.27%)

Transaction sent: 0x99294c2ce0e85b4a29fef4de6fd208aa89c9143469946e4ae70a702a3b027808
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  GempadLock.unlock confirmed   Block: 35   Gas used: 74253 (0.62%)

Transaction sent: 0x669d20b03b71931f90fa73696faaf26a0bc6a6b7f53b302263f45ee2e78ff55c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  GempadLock.unlock confirmed (Nothing to unlock)   Block: 36   Gas used: 32912 (0.27%)

Transaction sent: 0xbbfe67b0739c54815aaa5c56067e288ec7066d9a3076af4ced1bfa61bca285b1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  ERC20Mock.mint confirmed   Block: 37   Gas used: 50499 (0.42%)

Transaction sent: 0x4126eebb5ec980dd2a90613778f71adae69b32116d4dda1c630c46aa78383989
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  ERC20Mock.approve confirmed   Block: 38   Gas used: 43862 (0.37%)

Transaction sent: 0xadc0e7ec528a10a07017c192be4f6d8b29b82c4777f8ba0c94724a1c38e9dc63
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  GempadLock.multipleVestingLock confirmed   Block: 39   Gas used: 439310 (3.66%)

Transaction sent: 0x47ad6f1c3075635a8a70dfeeb2444bb4d9bf7f5da85b9d49ec80687a3c641195
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  GempadLock.unlock confirmed (You are not the owner of this lock)   Block: 40   Gas used: 30285 (0.25%)

Transaction sent: 0x38433386c99e160458e88c6692f9826c81819e571b911c8877b84bc81a3bc4d8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  GempadLock.unlock confirmed (Nothing to unlock)   Block: 41   Gas used: 41266 (0.34%)

Transaction sent: 0x5a364ad57a84a79b75266b40b116a91613e77b28df8f9c5b05f7479c61a64695
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  GempadLock.unlock confirmed   Block: 43   Gas used: 97750 (0.81%)

tests/test_gempad.py::test_unlock PASSED
```

```
tests/test_gempad.py::test_update_fee RUNNING
Transaction sent: 0x8c8c03a99c1c5eaa476e48db796e5d9e9e9e4b1a78540b6f24989a0744f515e4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  ERC20Mock.constructor confirmed   Block: 44   Gas used: 495827 (4.13%)
  ERC20Mock deployed at: 0x2c15A315610Bfa5248E4CbCbd693320e9D8E03Cc

Transaction sent: 0xb30c5a1ffa126cbfa0ea11582047aaaf2e464a11ce04c1586753d226c8d10789
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  GempadLock.constructor confirmed   Block: 45   Gas used: 5149710 (42.91%)
  GempadLock deployed at: 0xe692Cf21B12e0B2717C4bF647F9768Fa58861c8b

Transaction sent: 0x932505864ceadb14154d9d1f9708416c8424d789e3605134f46c9a03431cb53d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  GempadLock.updateFee confirmed (reverted)   Block: 46   Gas used: 23357 (0.19%)

Transaction sent: 0xf1dae0cb832e8f401c3a4c3fc8ebdfe5c5bde4d2380b3b27c15c43598783e6c5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  GempadLock.initialize confirmed   Block: 47   Gas used: 70389 (0.59%)

Transaction sent: 0xe554321ebf55c23ffb89925fe88854c98f373af1e0d6558a8213c1670bd3fcd3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  GempadLock.updateFee confirmed   Block: 48   Gas used: 125634 (1.05%)

Transaction sent: 0x22745850cee0723aa0777e481145423affed0600ecd28f4b10fa7df333b15eca
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  ERC20Mock.mint confirmed   Block: 49   Gas used: 65511 (0.55%)

Transaction sent: 0x6fa506d51a3ce23804650507ce0d43699695efd8db7ea99cfbc7e11032003e8d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  ERC20Mock.approve confirmed   Block: 50   Gas used: 43862 (0.37%)

Transaction sent: 0xcef87840c9fc2e8c5604f6051cd8756742d49be4f47c9869b0369a63b21e9abb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  GempadLock.multipleLock confirmed (Not enough funds for fees)   Block: 51   Gas used: 36220 (0.30%)

Transaction sent: 0xbfa30af3f7487a6686affae468cde4b895288f8823703ca086cc8f40b8bcad98
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 20
  GempadLock.multipleLock confirmed   Block: 52   Gas used: 481111 (4.01%)

tests/test_gempad.py::test_update_fee PASSED
tests/test_gempad.py::test_update_availability_for_nft RUNNING
Transaction sent: 0xed7e7c5f5bc65e3c68739a7095e9c0a803f63e0d0e5d3dcb3d98e55d86d2650c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  GempadLock.constructor confirmed   Block: 53   Gas used: 5149710 (42.91%)
  GempadLock deployed at: 0xFbD588c72B438faD4Cf7cD879c8F730Faa213Da0

Transaction sent: 0x9e79bd7ea77ce55a80beb13193efcb3aa1b5aa5a86abe3411fe126c7bd79a650
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 21
  GempadLock.updateAvailabilityForNFT confirmed (reverted)   Block: 54   Gas used: 22904 (0.19%)

Transaction sent: 0x15cd4be57773cfc26c6a55dad7b9ae3ff131f61fae8bb6204de03500e5944407
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  GempadLock.initialize confirmed   Block: 55   Gas used: 70389 (0.59%)

Transaction sent: 0xbf1fe3ab8c2e5e3dd978d0e52d2816d0ea111c6558122043d5af87c5bc653671
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  GempadLock.updateAvailabilityForNFT confirmed   Block: 56   Gas used: 46049 (0.38%)

Transaction sent: 0x5410c579b0b4c0443fc77697aefe2d3f5b57b643d47cac3d658db327aaca1086
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 20
  GempadLock.updateAvailabilityForNFT confirmed (the same)   Block: 57   Gas used: 23922 (0.20%)

tests/test_gempad.py::test_update_availability_for_nft PASSED
```

Annexes

Testing code:

Test GemPadLock contract:

```
from brownie import (
    reverts,
)


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


from scripts.deploy import (
    deploy_erc,
    deploy_gempad
)


def test_multiple_lock(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)


    token = deploy_erc(owner, "token", "TKN")
```

```
    gempad = deploy_gempad(owner)


    with reverts("Length mismatch"):
        gempad.multipleLock(
            [other, extra], token.address, False, [1e18],
            get_timestamp(1), "no_description", "no_metadata",
            token.address, other,
            {"from": other})


    with reverts("TGE date should be set in the future"):
        gempad.multipleLock(
            [other], token.address, False, [1e18],
            1, "no_description", "no_metadata",
            token.address, other,
            {"from": other})


    with reverts("Invalid token"):
        gempad.multipleLock(
            [other], ZERO_ADDRESS, False, [1e18],
            get_timestamp(1), "no_description", "no_metadata",
            token.address, other,
            {"from": other})


    with reverts("The amount cannot be zero"):
        gempad.multipleLock(
            [other], token.address, False, [0],
            get_timestamp(1), "no_description", "no_metadata",
            token.address, other,
```

```
        {"from": other})


token.mint(other, 5e18)
token.approve(gempad.address, 2e18, {"from": other})
tx = gempad.multipleLock(
        [other], token.address, False, [1e18],
        get_timestamp(1), "no_description", "no_metadata",
        token.address, other,
        {"from": other})


assert tx.events['LockAdded'][0]['id'] == 0
assert tx.events['LockAdded'][0]['owner'] == other
assert tx.events['LockAdded'][0]['metaData'] == "no_metadata"
assert tx.events['LockAdded'][0]['referrer'] == other
assert tx.events['LockAdded'][0]['locker'] == other


token.mint(extra, 5e18)
token.approve(gempad.address, 1e18, {"from": extra})
tx = gempad.multipleLock(
        [extra], token.address, False, [1e18],
        get_timestamp(1), "no_description", "no_metadata",
        token.address, other,
        {"from": extra})


assert tx.events['LockAdded'][0]['id'] == 1
assert tx.events['LockAdded'][0]['owner'] == other
assert tx.events['LockAdded'][0]['metaData'] == "no_metadata"
assert tx.events['LockAdded'][0]['referrer'] == other
```

```
    assert tx.events['LockAdded'][0]['locker'] == extra


    with reverts("This token is not the project token"):
        gempad.multipleLock(
            [other], token.address, False, [1e18],
            get_timestamp(1), "no_description", "no_metadata",
            extra, other,
            {"from": other})



def test_multiple_vesting_lock(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)


    token = deploy_erc(owner, "token", "TKN")
    gempad = deploy_gempad(owner)


    with reverts("Invalid cycle"):
        gempad.multipleVestingLock(
            [other], [1e18], token.address, False,
            get_timestamp(1), 500000, 0, 500000,
            "no_description", "no_metadata", token.address, other,
            {"from": other})


    with reverts("Invalid bips for TGE"):
        gempad.multipleVestingLock(
```

```
        [other], [1e18], token.address, False,
        get_timestamp(1), 0, 10000, 500000,
        "no_description", "no_metadata", token.address, other,
        {"from": other})

with reverts("Invalid bips for cycle"):
    gempad.multipleVestingLock(
        [other], [1e18], token.address, False,
        get_timestamp(1), 500000, 10000, 0,
        "no_description", "no_metadata", token.address, other,
        {"from": other})

with reverts("Sum of TGE bps and cycle should be less than 10000"):
    gempad.multipleVestingLock(
        [other], [1e18], token.address, False,
        get_timestamp(1), 500000, 10000, 600000,
        "no_description", "no_metadata", token.address, other,
        {"from": other})

token.mint(other, 5e18)
token.approve(gempad.address, 2e18, {"from": other})
tx = gempad.multipleVestingLock(
        [other], [1e18], token.address, False,
        get_timestamp(1), 500000, 10000, 500000,
        "no_description", "no_metadata", token.address, other,
        {"from": other})
assert tx.events['LockAdded'][0]['id'] == 0
assert tx.events['LockAdded'][0]['owner'] == other
```

```
    assert tx.events['LockAdded'][0]['metaData'] == "no_metadata"
    assert tx.events['LockAdded'][0]['referrer'] == other
    assert tx.events['LockAdded'][0]['locker'] == other

    token.mint(extra, 5e18)
    token.approve(gempad.address, 1e18, {"from": extra})
    tx = gempad.multipleVestingLock(
            [other], [1e18], token.address, False,
            get_timestamp(1), 500000, 10000, 500000,
            "no_description", "no_metadata", token.address, other,
            {"from": extra})
    assert tx.events['LockAdded'][0]['id'] == 1
    assert tx.events['LockAdded'][0]['owner'] == other
    assert tx.events['LockAdded'][0]['metaData'] == "no_metadata"
    assert tx.events['LockAdded'][0]['referrer'] == other
    assert tx.events['LockAdded'][0]['locker'] == extra



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

    token = deploy_erc(owner, "token", "TKN")
    gempad = deploy_gempad(owner)
```

```
    token.mint(other, 5e18)
    token.approve(gempad.address, 2e18, {"from": other})
    tx = gempad.multipleLock(
            [other], token.address, False, [1e18],
            get_timestamp(1), "no_description", "no_metadata",
            token.address, other,
            {"from": other})
    lock_id = tx.events['LockAdded'][0]['id']


    with reverts("Invalid lock ID"):
        gempad.unlock(1, {"from": extra})
    with reverts("You are not the owner of this lock"):
        gempad.unlock(lock_id, {"from": extra})
    with reverts("The lock is not unlocked yet"):
        gempad.unlock(lock_id, {"from": other})


    increase_timestamp(DAY_TIMESTAMP * 2)


    tx = gempad.unlock(lock_id, {"from": other})
    assert tx.events['Transfer'][0]['from'] == gempad.address
    assert tx.events['Transfer'][0]['to'] == other
    assert tx.events['Transfer'][0]['value'] == 1e18
    assert tx.events['LockRemoved'][0]['id'] == lock_id


    with reverts("Nothing to unlock"):
        gempad.unlock(lock_id, {"from": other})


    token.mint(extra, 5e18)
```

```
    token.approve(gempad.address, 1e18, {"from": extra})
    tx = gempad.multipleVestingLock(
            [another], [1e18], token.address, False,
            get_timestamp(1), 500000, 10000, 500000,
            "no_description", "no_metadata", token.address, extra,
            {"from": extra})
    lock_id = tx.events['LockAdded'][0]['id']
    with reverts("You are not the owner of this lock"):
        gempad.unlock(lock_id, {"from": extra})
    with reverts("Nothing to unlock"):
        gempad.unlock(lock_id, {"from": another})


    increase_timestamp(DAY_TIMESTAMP * 2)


    tx = gempad.unlock(lock_id, {"from": another})
    assert tx.events['Transfer'][0]['from'] == gempad.address
    assert tx.events['Transfer'][0]['to'] == another
    assert tx.events['Transfer'][0]['value'] == 1e18
    assert tx.events['LockRemoved'][0]['id'] == lock_id


def test_update_fee(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)


    token = deploy_erc(owner, "token", "TKN")
    gempad = deploy_gempad(owner)
```

```
with reverts():
    gempad.updateFee(1000, 1000, 1000, 1000, 1000, {"from": extra})


gempad.initialize({"from": owner})
assert gempad.fee()[0] == 0
assert gempad.fee()[1] == 0
assert gempad.fee()[2] == 0
assert gempad.fee()[3] == 0
assert gempad.fee()[4] == 0
tx = gempad.updateFee(1000, 2000, 3000, 4000, 5000, {"from": owner})
assert gempad.fee()[0] == 1000
assert gempad.fee()[1] == 2000
assert gempad.fee()[2] == 3000
assert gempad.fee()[3] == 4000
assert gempad.fee()[4] == 5000


token.mint(other, 5e18)
token.approve(gempad.address, 2e18, {"from": other})
with reverts("Not enough funds for fees"):
    gempad.multipleLock(
        [other], token.address, False, [1e18],
        get_timestamp(1), "no_description", "no_metadata",
        token.address, other,
        {"from": other})
tx = gempad.multipleLock(
        [other], token.address, False, [1e18],
        get_timestamp(1), "no_description", "no_metadata",
```

```
            token.address, other,
            {"from": other, "value": 5000})
    assert tx.events['LockAdded'][0]['id'] == 0
    assert tx.events['LockAdded'][0]['owner'] == other
    assert tx.events['LockAdded'][0]['metaData'] == "no_metadata"
    assert tx.events['LockAdded'][0]['referrer'] == other
    assert tx.events['LockAdded'][0]['locker'] == other


def test_update_availability_for_nft(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)


    random_nft = "0x29bc8ba7e29dF4868c35Cafb4caeE7BD3cEe9CEE"
    gempad = deploy_gempad(owner)


    with reverts():
        gempad.updateAvailabilityForNFT(random_nft, True, {"from": other})
    gempad.initialize({"from": owner})
    assert gempad.isAvailableNFT(random_nft) == False
    tx = gempad.updateAvailabilityForNFT(random_nft, True, {"from": owner})
    assert tx.events['NFTAvailableUpdated'][0]['nft'] == random_nft
    assert tx.events['NFTAvailableUpdated'][0]['isAvailable'] == True
    assert gempad.isAvailableNFT(random_nft) == True


    with reverts("the same"):
        gempad.updateAvailabilityForNFT(random_nft, True, {"from": owner})
```

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	0					
Medium	3					
Low	6					
Informational	0					

Assessment Results

Score Results

Review	Score
Global Score	**70/100**
Assure KYC	Not completed
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 token contract for the GemPadLock project, we inform you that the contract has not met the required standards and reported medium/low vulnerabilities must be reviewed prior to deployment.

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.