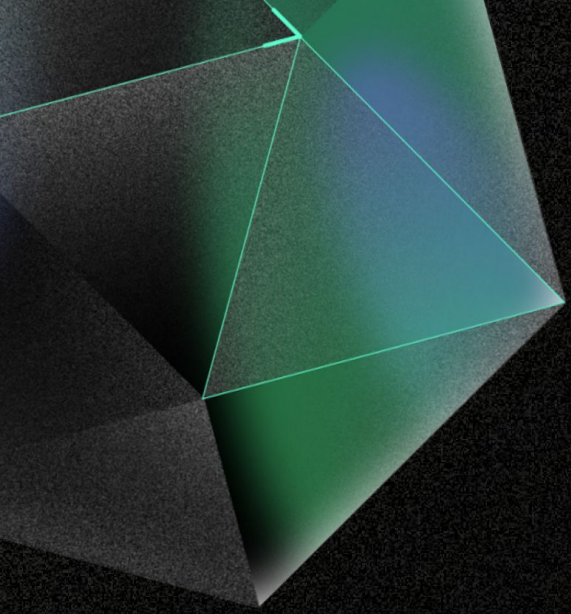
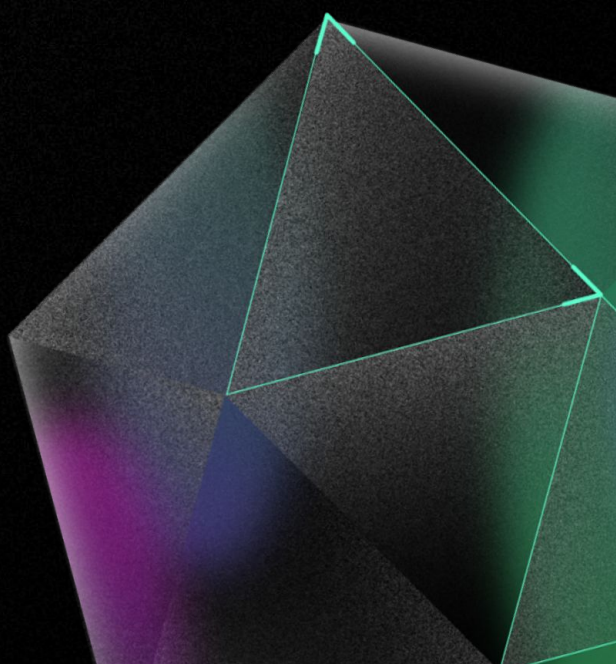


hexens

SMART CONTRACT AUDIT REPORT FOR NIMBUS PLATFORM

16.12.2021



Contents

Summary

Severity	Number of Findings
High	3
Medium	1
Low	1
Informational	0

Total: 5

Scope

The analyzed contracts are located in the following repository:
https://github.com/nimbusplatformorg/nim-smartcontract/contracts/contracts_BSC/dApps/NFTTokens/TikToken.sol
(commit 1b15ac00e0c3b5c260ac44d98ad8420b894a8a1a).
https://github.com/nimbusplatformorg/nim-smartcontract/contracts/contracts_BSC/dApps/P2P/NimbusP2P_V2.sol
(commit 84475278eb4c5a25d2a943bc7bdbb9e07edb53bc)

Description

This audit covered the recent changes made to the Nimbus Platform protocol: TikToken.sol and NimbusP2P_V2.sol contracts were added to the project.

Test cases

Below are the testing results. The test cases corresponding to the vulnerabilities are highlighted in red.

```
  Nimbus
    Deployment
      ✓ deploying the tokens (542ms)
      ✓ deploying the factory and pair (186ms)
      ✓ deploying the router and lprewards (174ms)
      ✓ deploying the referral program (225ms)
      ✓ minting the tokens
    Staking
      ✓ prepare the pairs (397ms)
      ✓ add liquidity for the NBU router (184ms)
      ✓ set up NBU soft lprewards staking (66ms)
      ✓ send money to the soft NBU lprewards staking (144ms)
      ✓ add liquidity for the GNBU router (169ms)
      ✓ add liquidity for the NBU/GNBU router (110ms)
      ✓ set up GNBU soft lprewards staking (83ms)
      ✓ send money to the soft GNBU lprewards staking for
someone (118ms)
      ✓ fund the reward tokens for staking (41ms)
    Lending
      ✓ deploy the loan token (142ms)
      ✓ deploy the protocol (310ms)
      ✓ someone deposits BNB
      ✓ [!] no one can deposit (60ms)
      ✓ manipulate loan (147ms)
      ✓ fund the loan tokens
    TikToken
      ✓ deploy NFT token (76ms)
      ✓ buy some NFT (1364ms)
      ✓ [!] cannot burn the NFTs (97ms)
      ✓ cannot burn someone else's NFT
      ✓ someone adds liquidity BNB (88ms)
      ✓ emulate the interest settling
      ✓ re-entrancy fails because of the staking nonces
(929ms)
    P2P
```

```
✓ deploy NimbusP2P_V2
      ✓ reset WBNB address (custom for audit)
      ✓ cannot buy bnb for bnb
      ✓ cannot buy for 0
      ✓ offer NBU for NBU_WETH
      ✓ cancel trade
      ✓ cannot support a dead trade
      ✓ again offer NBU for NBU_WBNB
      ✓ a hacker cannot get free NBU
      ✓ [!] offer NFT as BEP (85ms)
      ✓ allow any NFT
      ✓ cannot offer NBU as NFT


  39 passing (1m)
```

Weaknesses

This section contains the list of discovered weaknesses.

1. LoanTokenLogicWbnb cannot receive BNB

Severity: High

Resolution: reimplement the **withdraw** function in **NBU_WETH** using **transfer** instead of **call**.

Description:

The method **withdraw** in the file **contracts_BSC/Swaps/NBU_WBNB.sol** uses the address call to send BNB instead of transferring.

This makes it impossible for some of the contracts to receive BNB. For example, the fallback method of the **LoanToken** contract does not support such calls.

Below is the method's code starting from the line 19 in the file **LoantToken.sol**:

```
function()
    external
    payable
{
    if (gasleft() <= 2300) {
        return;
    }

    address target = target_;
    bytes memory data = msg.data;
    assembly {
```

```
        let result := delegatecall(gas, target, add(data, 0x20),
mload(data), 0, 0)
        let size := returndatasize
        let ptr := mload(0x40)
        returndatacopy(ptr, 0, size)
        switch result
        case 0 { revert(ptr, size) }
        default { return(ptr, size) }
    }
}
```

2. Re-entrancy is possible in TikToken

Severity: High

Resolution: Add the locking mechanism to prevent re-entrancy.

Description:

There's a possibility of a re-entrancy attack on the **TikToken** contract since: there's no locking mechanism in the **withdrawUserRewards** and **burnTikToken** methods.

The burnTikToken method could be used for re-entrancy due to the BNB withdrawal, however, right now, the **lpStakingBnbNbu.withdraw** call prevents it due to the impossibility of the nonce reuse.

During future refactoring or updates, this issue can become critical and may lead to the loss of funds.

Below is the method's code starting from the line 544 in the file **TikToken.sol**:

```
    function burnTikToken(uint tokenId) external {
        require(_owners[tokenId] == msg.sender, "TikToken: Not token
owner");
        UserSupply storage userSupply = tikSupplies[tokenId];
        require(userSupply.IsActive, "TikToken: Token not active");
        (uint lpBnbNbuUserRewards, uint lpBnbGnbuUserRewards, ) =
getTokenRewardsAmounts(tokenId);

        if(lpBnbNbuUserRewards + lpBnbGnbuUserRewards > 0) {
            _withdrawUserRewards(tokenId, lpBnbNbuUserRewards,
lpBnbGnbuUserRewards);
        }

        lpStakingBnbNbu.withdraw(userSupply.NbuBnbStakeNonce);
        swapRouter.removeLiquidityBNB(address(nbuToken),
userSupply.NbuBnbLpAmount, 0, 0,  msg.sender, block.timestamp);

        lpStakingBnbGnbu.withdraw(userSupply.GnbuBnbStakeNonce);
        swapRouter.removeLiquidityBNB(address(gnbuToken),
userSupply.GnbuBnbLpAmount, 0, 0, msg.sender, block.timestamp);

        lendingContract.burnToBnb(msg.sender,
userSupply.LendedITokenAmount);

        transferFrom(msg.sender, address(0x1), tokenId);
        userSupply.IsActive = false;

        emit BurnTikToken(tokenId);
    }
```

3. Financial Inconsistency of LoanToken

Severity: High

Resolution: Consider adding extra checks to prevent DoS in TikToken.

Description:

During the NFT burn, the calculation of the **lendedUserRewards** value in the **getTokenRewardsAmounts** function leads to the integer underflow and makes the withdrawal impossible.

The reason for this is that when lending token price drops below the price at the moment of **TikToken** minting, the rewards will become negative.

Below is the method's code starting from the line 570 in the file **TikToken.sol**:

```
function getTokenRewardsAmounts(uint tokenId) public view returns (uint lpBnbNbuUserRewards, uint lpBnbGnbuUserRewards, uint lendedUserRewards) {
    UserSupply memory userSupply = tikSupplies[tokenId];
    require(userSupply.IsActive, "TikToken: Not active");
lpBnbNbuUserRewards = (_balancesRewardEquivalentBnbNbu[tokenId] * ((block.timestamp - weightedStakeDate[tokenId]) * 100)) / (100 * rewardDuration);
    lpBnbGnbuUserRewards = (_balancesRewardEquivalentBnbGnbu[tokenId] * ((block.timestamp - weightedStakeDate[tokenId]) * 100)) / (100 * rewardDuration);
```

```
 lpBnbGnbuUserRewards =
(_balancesRewardEquivalentBnbGnbu[tokenId] * ((block.timestamp -
weightedStakeDate[tokenId]) * 100)) / (100 * rewardDuration);
        lendedUserRewards = (userSupply.LendedITokenAmount *
lendingContract.tokenPrice() / 1e18) -
userSupply.LendedBNBAmount;
    }
```

4. Type Confusion in NimbusP2P_V2

Severity: Medium

Resolution: require the **percent** value to be lower than a certain threshold.

Description:

There's a type confusion due to the backward compatibility of BEP721 and BEP20 in the **NimbusP2P_V2** contract:: one can create an NFT deal even if NFT deals are not enabled.

Just call the **createTradeEIP20ToEIP20** method with the NFT asset address and the token id instead of the amount.

The opposite is not possible because there's no function selector for the method **safeTransferFrom** in the BEP20 contracts.

Below is the method's code starting from the line 195 in the file **NimbusP2P_V2.sol**:

```
function createTradeEIP20ToEIP20(address proposedAsset, uint
proposedAmount, address askedAsset, uint askedAmount, uint
deadline) external returns (uint tradeId) {
        require(Address.isContract(proposedAsset) &&
Address.isContract(askedAsset), "NimbusP2P_V2: Not contracts");
        require(proposedAmount > 0, "NimbusP2P_V2: Zero amount not
allowed");
        TransferHelper.safeTransferFrom(proposedAsset, msg.sender,
address(this), proposedAmount);
        tradeId = _createTradeSingle(proposedAsset, proposedAmount,
0, askedAsset, askedAmount, 0, deadline, false);
    }
```

5. Misprint in the error message in TikToken

Severity: Low

Resolution: fix the misprint.

Description:

The error message "TikToken: min purchase is too low" in the **buyTikToken** method could be misleading.


hexens