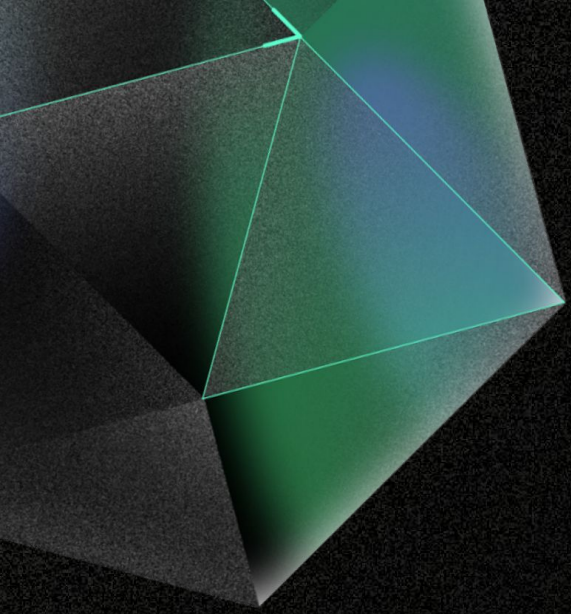
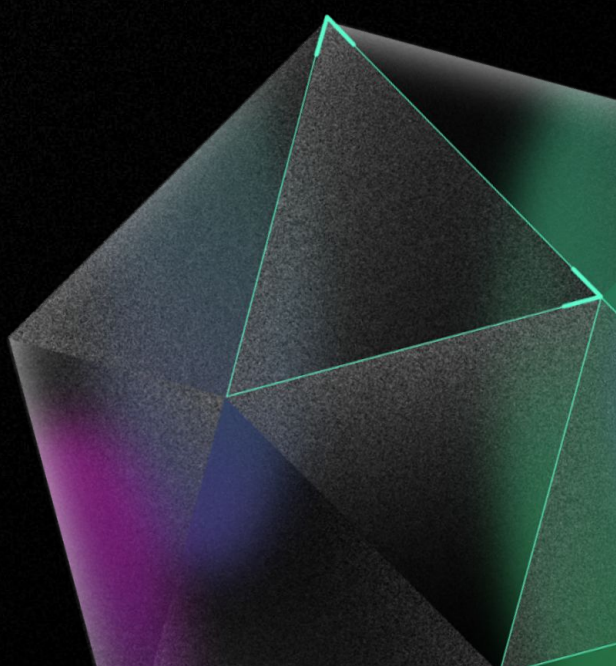


hexens

SMART CONTRACT AUDIT REPORT FOR NIMBUS PLATFORM

27.09.2021



Contents

Summary

Severity	Number of Findings
High	1
Medium	1
Low	5
Informational	0

Total: 7

Scope

The analyzed smart contracts are located in the following repositories:

https://github.com/nimbusplatformorg/nim-smartcontract/blob/master/contracts/contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol

https://github.com/nimbusplatformorg/nim-smartcontract/blob/master/contracts/contracts_BSC/InitialAcquisition/NimbusInitialAcquisitionReferral_3-0.sol

https://github.com/nimbusplatformorg/nim-smartcontract/blob/master/contracts/contracts_BSC/NimbusCore/NimbusVesting.sol

Test cases

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

```
Nimbus
  Deployment
    ✓ deploying the tokens (708ms)
    ✓ deploying the factory and pair (254ms)
    ✓ deploying the router and lprewards (313ms)
    ✓ deploying the referral program (338ms)
    ✓ minting the tokens
  Vesting 2.0
    ✓ deploying vesting (63ms)
    ✓ random guy cannot vest
    ✓ add vesters
    ✓ [!] bad vesting for user
    ✓ good vesting for user
    ✓ [!] unvesting broken
    ✓ allow anyone unvest
    ✓ another vesting for user
    ✓ put tokens into the vesting contract (99ms)
    ✓ now anyone can unvest (42ms)
    ✓ random guy cannot add vesters
    ✓ vesting with type
  Marketing 3.0
    ✓ deploying NimbusReferralProgramMarketing (137ms)
    ✓ add the regional managers
    ✓ add a head of location
    ✓ add the registrators
    ✓ add as a registrator
    ✓ update vesting
    ✓ transfer tokens (43ms)
    ✓ [!] registering malicious managers in marketing (97ms)
    ✓ registering in marketing by sponsor address (54ms)
    ✓ deploying NimbusInitialAcquisition 3.0 (450ms)
    ✓ add as a vester
    ✓ update swap token
    ✓ update swap threshold
    ✓ add as a vester
```

```
✓ update referral program
      ✓ only owner can toggle weighted
      ✓ set marketing address
      ✓ add as a registrator
      ✓ add as an allowed contract
      ✓ get token amount
      ✓ create a staking pool (67ms)
      ✓ someone invests (51ms)
      ✓ add the staking pool
      ✓ add allowed tokens
      ✓ buy for exact bnb (137ms)
      ✓ [!] turnover double recording
      ✓ owner updates qualifications
      ✓ only owner can import turnovers
      ✓ rewards calculated correctly (73ms)
      ✓ claim rewards (41ms)
      ✓ register without sponsor
      ✓ again buy for exact bnb (134ms)


 49 passing (1m)
```

Weaknesses

This section contains the list of discovered weaknesses.

1. Incorrect turnover calculation in NimbusReferralProgramMarketing

Severity: High

Resolution: avoid double calculation of the manager's turnover.

Description:

The method **_updateReferralProfitAmount** in the file **contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol** updates the turnover values for the user, their sponsors, and the appropriate regional manager. But it doesn't properly account for a case when there's a head of location in the referral chain.

This leads to the double recording of the regional manager's turnover whenever a head of location is encountered in the referral chain.

Below is the method's code starting from the line 301 in the file **NimbusReferralProgramMarketing.sol**:

```
function _updateReferralProfitAmount(address user, uint amount,
uint line) internal {
    if (line == 0) {
        userPersonalTurnover[user] += amount;
        emit UpdateReferralProfitAmount(user, amount, line);
        address userSponsor =
rpUsers.userSponsorAddressByAddress(user);
        if (isHeadOfLocation[user]) {
```

```
 headOfLocationTurnover[user] += amount;
            address regionalManager =
headOfLocationRegionManagers[user];
            regionalManagerTurnover[regionalManager] += amount;
        } else if (isRegionManager[user]) {
            regionalManagerTurnover[user] += amount;
            return;
        } else {
            _updateReferralProfitAmount(userSponsor, amount, 1);
        }
    } else {
        userStructureTurnover[user] += amount;
        emit UpdateReferralProfitAmount(user, amount, line);
        if (isHeadOfLocation[user]) {
            headOfLocationTurnover[user] += amount;
            address regionalManager =
headOfLocationRegionManagers[user];
            regionalManagerTurnover[regionalManager] += amount;
        } else if (isRegionManager[user]) {
            regionalManagerTurnover[user] += amount;
            return;
        }

        if (line >= REFERRAL_LINES) {
            _updateReferralHeadOfLocationAndRegionalTurnover(user,
amount);
            return;
```

```
    }

        address userSponsor =
rpUsers.userSponsorAddressByAddress(user);
        if (userSponsor == address(0)) {
            _updateReferralHeadOfLocationAndRegionalTurnover(user,
amount);
            return;
        }

        _updateReferralProfitAmount(userSponsor, amount, ++line);
    }
}
```

2. Incorrect unvesting conditions in NimbusVesting

Severity: Medium

Resolution: replace *break* clause with *continue* clause.

Description:

The methods **_unvest, availableForUnvesting,** and **userUnvested** in the file **contracts_BSC/NimbusCore/NimbusVesting.sol** contains a loop which stops if the current vesting transaction's release date has not yet come.

This prevents the unvesting of all the later vesting transactions even if it's due.

Below is the method's code starting from the line 203 in the file **NimbusVesting.sol**:

```
function _unvest(address user) internal returns (uint unvested) {
    uint nonce = vestingNonces[user];
    require (nonce > 0, "NimbusVesting: No vested amount");
    for (uint i = 1; i <= nonce; i++) {
        VestingInfo memory vestingInfo = vestingInfos[user][i];
        if (vestingInfo.vestingAmount == vestingInfo.unvestedAmount)
continue;
        if (vestingInfo.vestingReleaseStartDate > block.timestamp)
break;
        uint toUnvest;
        if (vestingInfo.vestingSecondPeriod != 0) {
            toUnvest = (block.timestamp -
vestingInfo.vestingReleaseStartDate) * vestingInfo.vestingAmount /
vestingInfo.vestingSecondPeriod;
            if (toUnvest > vestingInfo.vestingAmount) {
                toUnvest = vestingInfo.vestingAmount;
            }
        } else {
            toUnvest = vestingInfo.vestingAmount;
        }
  uint totalUnvestedForNonce = toUnvest;
        toUnvest -= vestingInfo.unvestedAmount;
        unvested += toUnvest;
        vestingInfos[user][i].unvestedAmount = totalUnvestedForNonce;
    }
    require(unvested > 0, "NimbusVesting: Unvest amount is zero");
    vestingToken.safeTransfer(user, unvested);
    emit Unvest(user, unvested);
}
```

3. Dead code in NimbusReferralProgramMarketing

Severity: Low

Resolution: implement the setter for *userHeadOfLocations*.

Description:

The method _updateReferralHeadOfLocationAndRegionalTurnover in the file contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol checks the value of the element in the **userHeadOfLocations** mapping and exists if it's null.

This makes the function dead code because the **userHeadOfLocations** values are never set.

Below is the method's code starting from the line 347 in the file NimbusReferralProgramMarketing.sol:

```
function _updateReferralHeadOfLocationAndRegionalTurnover(address user, uint amount) internal {
    address headOfLocation = userHeadOfLocations[user];
    if (headOfLocation == address(0)) return;
    headOfLocationTurnover[headOfLocation] += amount;
    address regionalManager = headOfLocationRegionManagers[user];
    emit UpdateHeadOfLocationTurnover(headOfLocation, amount);
    if (regionalManager == address(0)) return;
    regionalManagerTurnover[regionalManager] += amount;
    emit UpdateRegionalManagerTurnover(regionalManager, amount);
  }
```

4. Misleading error message in NimbusReferralProgramMarketing

Severity: Low

Resolution: change the error message.

Description:

The method **addRegionalManager** in the file **contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol** reverts if the regional manager already exists.

However, the revert message says the opposite and therefore is misleading for the users.

Below is the method's code starting from the line 476 in the file **NimbusReferralProgramMarketing.sol**:

```
function addRegionalManager(address regionalManager) external
onlyOwner {
        require(!isRegionManager[regionalManager],
"NimbusReferralProgramMarketing: No such regional manager");
        regionalManagers.push(regionalManager);
        isRegionManager[regionalManager] = true;
        emit AddRegionalManager(regionalManager);
    }
```

5. Potential out-of-gas DoS in NimbusVesting and NimbusReferralProgramMarketing

Severity: Low

Resolution: implement a possibility for partial execution of critical functions such as unvesting and reward distribution.

Description:

The method **getUserRewards** in the file **contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol** and the method **_unvest** in the file **contracts_BSC/NimbusCore/NimbusVesting.sol** are critical for the business logic and contain unconstrained loops.

This may lead to the losses for the users whose funds have been blocked due to the out-of-gas during the loop execution.

Below is the method's code starting from the line 247 in the file **NimbusReferralProgramMarketing.sol**:

```
function getUserRewards(address user) public view returns (uint
userFixed, uint userVariable, uint potentialLevel) {
    require(rpUsers.userIdByAddress(user) > 0,
"NimbusReferralProgramMarketing: User not registered");
    uint qualificationLevel = userQualificationLevel[user];
    potentialLevel = _getUserPotentialQualificationLevel(user,
qualificationLevel);
    require(potentialLevel > qualificationLevel,
"NimbusReferralProgramMarketing: User level hasn't changed");
    //uint userTurnover = userTotalTurnover(user);
    uint[] memory userReferrals = rpUsers.getUserReferrals(user);
    if (userReferrals.length == 0) return (0, 0, potentialLevel);
```

```
address[] memory referralAddresses = new
address[](userReferrals.length);
    for (uint i; i < userReferrals.length; i++) {
        address referralAddress =
rpUsers.userAddressById(userReferrals[i]);
        referralAddresses[i] = referralAddress;
    }
userFixed = _getFixedRewardToBePaidForQualification(user,
referralAddresses, userTotalTurnover(user), qualificationLevel,
potentialLevel);
    userVariable =
_getVariableRewardToBePaidForQualification(referralAddresses,
userStructureTurnover[user], potentialLevel);
}
```

6. [NEW] Integer underflow in NimbusReferralProgramMarketing

Severity: Low

Resolution: check the value before subtraction.

Description:

The method **_getUserPotentialQualificationLevel** in the updated file **contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol** subtracts 1 from the potentially zero value (**qualificationsCount** variable).

This may lead to an unexpected revert of the transaction.

Below is the method's code starting from the line 360 in the file **NimbusReferralProgramMarketing.sol**:

```
function _getUserPotentialQualificationLevel(address user, uint
qualificationLevel) internal view returns (uint) {
    if (qualificationLevel >= qualificationsCount) return
qualificationsCount - 1;

    uint turnover = userTotalTurnover(user);
for (uint i = qualificationLevel; i < qualificationsCount; i++) {
        if (qualifications[i+1].TotalTurnover > turnover) {
            return i;
        }
    }
    return qualificationsCount - 1; //user gained max qualification
}
```

7. [NEW] Recording turnover for a 0 address in NimbusReferralProgramMarketing

Severity: Low

Resolution: check the address before recording turnover.

Description:

The method **_updateReferralHeadOfLocationAndRegionalTurnover** in the updated file **contracts_BSC/NimbusCore/NimbusReferralProgramMarketing.sol** doesn't check if the input address is 0. The address can be 0 if there's someone who registered with the default sponsor id in the chain, and, as a result, there's also the default user id (1000000001) without any sponsor.

This may lead to undesired behaviour because of the extra turnover recorded.

Below is the method's code starting from the line 360 in the file NimbusReferralProgramMarketing.sol:

```
function _updateReferralProfitAmount(address user, uint amount,
uint line, bool isRegionalAmountUpdated) internal {
    if (line == 0) {
        userPersonalTurnover[user] += amount;
        emit UpdateReferralProfitAmount(user, amount, line);
        address userSponsor =
rpUsers.userSponsorAddressByAddress(user);
        if (isHeadOfLocation[user]) {
            headOfLocationTurnover[user] += amount;
            address regionalManager =
headOfLocationRegionManagers[user];
            regionalManagerTurnover[regionalManager] += amount;
            isRegionalAmountUpdated = true;
        } else if (isRegionManager[user]) {
            regionalManagerTurnover[user] += amount;
            return;
        } else {
            _updateReferralProfitAmount(userSponsor, amount, 1,
isRegionalAmountUpdated);
        }
    } else {
        userStructureTurnover[user] += amount;
        emit UpdateReferralProfitAmount(user, amount, line);
        if (isHeadOfLocation[user]) {
            headOfLocationTurnover[user] += amount;
            address regionalManager =
headOfLocationRegionManagers[user];
            if (!isRegionalAmountUpdated) {
```

```
    regionalManagerTurnover[regionalManager] += amount;
                isRegionalAmountUpdated = true;
            }
        } else if (isRegionManager[user]) {
            if (!isRegionalAmountUpdated)
regionalManagerTurnover[user] += amount;
            return;
        }

        if (line >= REFERRAL_LINES && !isRegionalAmountUpdated) {
            _updateReferralHeadOfLocationAndRegionalTurnover(user,
amount);
            return;
        }

        address userSponsor =
rpUsers.userSponsorAddressByAddress(user);
        if (userSponsor == address(0) && !isRegionalAmountUpdated) {
            _updateReferralHeadOfLocationAndRegionalTurnover(user,
amount);
            return;
        }

        _updateReferralProfitAmount(userSponsor, amount, ++line,
isRegionalAmountUpdated);
    }
}
```

Conclusion

The discovered weaknesses may lead to the following threats:

1) Attacker can claim unjustly high bonus from referral system,
2) Logical bug that breaks function integrity,
3) DoS (Denial of Service) possibilities.


hexens