Competitive Security Assessment

ParaSpace V1.4 P2

Feb 17th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	ParaSpace V1.4 P2
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - 6ba3c430a24b9781fee5f0c0745862748655b36e • final commit - 6ba3c430a24b9781fee5f0c0745862748655b36e
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

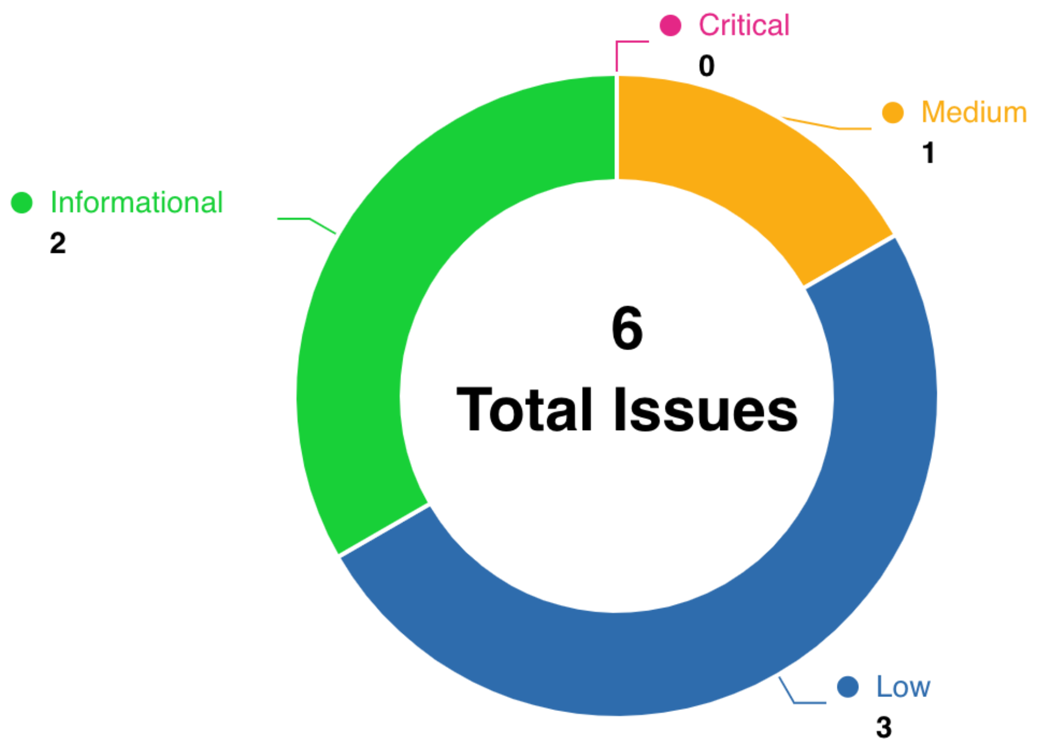
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**0**	**0**	**0**	**0**	**0**	**0**
Medium	**1**	**0**	**1**	**0**	**0**	**0**
Low	**3**	**0**	**2**	**0**	**0**	**1**
Informational	**2**	**0**	**1**	**0**	**0**	**1**

Audit Scope

File	Commit Hash
contracts/protocol/pool/PoolApeStaking.sol	6ba3c430a24b9781fee5f0c0745862748655b36e
contracts/protocol/tokenization/NTokenApeStaking.sol	6ba3c430a24b9781fee5f0c0745862748655b36e
contracts/protocol/tokenization/NTokenBAKC.sol	6ba3c430a24b9781fee5f0c0745862748655b36e
contracts/protocol/tokenization/libraries/ApeStakingLogic.sol	6ba3c430a24b9781fee5f0c0745862748655b36e

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PSV-1	Missing repeatability check for `tokenIds` & `_nftPairs` in `PoolApeStaking` contract `claimBAKC` & `withdrawBAKC` & `claimPairedApeAndCompound` function	Logical	Low	Acknowledged	w2ning
PSV-2	New BAKC owners can lock the staked APECoin of BAYC/MAYC owners in the contract	Logical	Medium	Acknowledged	thereksfour

PSV-3	**PairedApe stakers can prevent other users from calling claimPairedApeAndCompound to earn compoundFee**	**Logical**	**Low**	**Acknowledged**	**thereksfour**
PSV-4	**Potential Reentrancy risk in `PoolApeStaking` contract `repayAndSupply` function**	**Logical**	**Low**	**Declined**	**w2ning**
PSV-5	**`ApeStakingLogic::getTokenIdStakingAmount` should consider the BAKC reward APE token**	**Logical**	**Informational**	**Acknowledged**	**comcat**
PSV-6	**`PoolApeStaking::_validateBAKCOwnerAndTransfer` Gas optimization**	**Gas Optimization**	**Informational**	**Declined**	**comcat**

PSV-1:Missing repeatability check for `tokenIds` & `_nftPairs` in `PoolApeStaking` contract `claimBAKC` & `withdrawBAKC` & `claimPairedApeAndCompound` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/pool/PoolApeStaking.sol#L139 • code/contracts/protocol/pool/PoolApeStaking.sol#L193 • code/contracts/protocol/pool/PoolApeStaking.sol#L446	Acknowledged	w2ning

Code

```
139:    function withdrawBAKC(

193:    function claimBAKC(

446:    function claimPairedApeAndCompound(
```

Description

w2ning : Missing repeatability check for array `tokenIds` & `_nftPairs` in `PoolApeStaking` contract `claimBAKC` & `withdrawBAKC` & `claimPairedApeAndCompound` function.

A malicious user can pass in a array with the same elements, When the NFT is transferred back to the user, the user can transfer the NFT to the 'bakcContract' again through the ERC721 default hook function `onERC721Received`, and the cycle can proceed normally.

Recommendation

w2ning : Check the repeatability of elements in `tokenIds` & `nftPaires` array

Consider below fix in the `PoolApeStaking.claimApeAndCompound()` function

```
function claimApeAndCompound(
    address nftAsset,
    address[] calldata users,
    uint256[][] calldata tokenIds
) external nonReentrant {

  // fix: Check the repeatability of elements in `tokenIds` array
  for (uint256 i = 0; i < tokenIds.length; i++) {

    for (uint256 j = 0; j < tokenIds.length; j++) {

      if (i != j){
          require(
              tokenIds[i] != tokenIds[j],
              "TokenId can not be same"
          );

        }
    }
  }

   ...
}
```

Client Response

The protocol will not suffer because of this issue. so we don't need to waste gas to do this checking.

PSV-2:New BAKC owners can lock the staked APECoin of BAYC/MAYC owners in the contract

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/pool/PoolApeStaking.sol#L765-L781	Acknowledged	thereksfour

Code

```
    function _validateBAKCOwnerAndTransfer(
        ApeStakingLocalVars memory localVar,
        uint256 tokenId,
        address userAddress
    ) internal returns (address bakcOwner) {
        bakcOwner = localVar.bakcContract.ownerOf(tokenId);
        require(
            (userAddress == bakcOwner) ||
                (userAddress == INToken(localVar.bakcNToken).ownerOf(tokenId)),
            Errors.NOT_THE_BAKC_OWNER
        );
        localVar.bakcContract.safeTransferFrom(
            bakcOwner,
            localVar.xTokenAddress,
            tokenId
        );
    }
```

Description

thereksfour : In ApeCoinStaking, either BAYC/MAYC or BAKC owner can unstake BAYC/MAYC+BAKC, in which case if the NFT owner is different, the staked APECoin will be sent to the BAYC/MAYC owner and the reward APECoin will be sent to the BAKC owner.

```
            if (mainTokenOwner != msg.sender) {
                if (bakcOwner != msg.sender) revert NeitherTokenInPairOwnedByCaller();
            }
...
            if (pair.isUncommit) {
                uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID, position, bakcOwner);
                mainToBakc[mainTypePoolId][pair.mainTokenId] = PairingStatus(0, false);
                bakcToMain[pair.bakcTokenId][mainTypePoolId] = PairingStatus(0, false);
                emit ClaimRewardsPairNft(msg.sender, rewardsToBeClaimed, mainTypePoolId,
pair.mainTokenId, pair.bakcTokenId);
            }
            uint256 finalAmountToWithdraw = pair.isUncommit ? position.stakedAmount: pair.amount;
            _withdraw(BAKC_POOL_ID, position, finalAmountToWithdraw);
            apeCoin.transfer(mainTokenOwner, finalAmountToWithdraw);
```

PoolApeStaking contract still accepts BAKC (not nBAKC) for staking.

```
    function _validateBAKCOwnerAndTransfer(
        ApeStakingLocalVars memory localVar,
        uint256 tokenId,
        address userAddress
    ) internal returns (address bakcOwner) {
        bakcOwner = localVar.bakcContract.ownerOf(tokenId);
        require(
            (userAddress == bakcOwner) ||
                (userAddress == INToken(localVar.bakcNToken).ownerOf(tokenId)),
            Errors.NOT_THE_BAKC_OWNER
        );
        localVar.bakcContract.safeTransferFrom(
            bakcOwner,
            localVar.xTokenAddress,
            tokenId
        );
    }
```

Consider the following scenario: Where alice calls PoolApeStaking.borrowApeAndStake with nBAYC#1111 and BAKC#2222 for staking, deposits 1000 APECoin And alice later sells BAKC#2222 to bob. At this point, if bob calls ApeCoinStaking.withdrawBAKC directly with BAKC#2222, the rewarded APECoin will be sent to bob, while the 1000 staked APECoin will be sent to nBAYC, and alice cannot call PoolApeStaking.withdrawBAKC to withdraw the staked APECoin.

Recommendation

thereksfour : Consider accepting only nBAKC for staking.

Client Response

Currently we rescue staked ape coin for user from nBAYC/BAKC if this issue happened. We'll only support nBACK for pair staking later.

PSV-3:PairedApe stakers can prevent other users from calling claimPairedApeAndCompound to earn compoundFee

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/pool/PoolApeStaking.sol#L446-L502	Acknowledged	thereksfour

Code

```
    function claimPairedApeAndCompound(
        address nftAsset,
        address[] calldata users,
        ApeCoinStaking.PairNft[][] calldata _nftPairs
    ) external nonReentrant {
        require(
            users.length == _nftPairs.length,
            Errors.INCONSISTENT_PARAMS_LENGTH
        );
        DataTypes.PoolStorage storage ps = poolStorage();

        ApeStakingLocalVars memory localVar = _compoundCache(
            ps,
            nftAsset,
            users.length
        );

        for (uint256 i = 0; i < _nftPairs.length; i++) {
            localVar.transferredTokenOwners = new address[](
                _nftPairs[i].length
            );
            for (uint256 j = 0; j < _nftPairs[i].length; j++) {
                require(
                    users[i] ==
                        INToken(localVar.xTokenAddress).ownerOf(
                            _nftPairs[i][j].mainTokenId
                        ),
                    Errors.NOT_THE_OWNER
                );

                localVar.transferredTokenOwners[
                        j
                    ] = _validateBAKCOwnerAndTransfer(
                    localVar,
                    _nftPairs[i][j].bakcTokenId,
                    users[i]
                );
            }

            INTokenApeStaking(localVar.xTokenAddress).claimBAKC(
                _nftPairs[i],
                address(this)
            );

            for (uint256 index = 0; index < _nftPairs[i].length; index++) {
                localVar.bakcContract.safeTransferFrom(
                    localVar.xTokenAddress,
                    localVar.transferredTokenOwners[index],
                    _nftPairs[i][index].bakcTokenId
                );
            }

            _addUserToCompoundCache(ps, localVar, i, users[i]);
        }

        _compoundForUsers(ps, localVar, users);
    }
```

Description

thereksfour : PoolApeStaking allows users to call claimApeAndCompound/claimPairedApeAndCompound for Ape staker to earn compoundFee.

```
    function _compoundForUsers(
        DataTypes.PoolStorage storage ps,
        ApeStakingLocalVars memory localVar,
        address[] calldata users
    ) internal {
        APE_COMPOUND.deposit(
            address(this),
            localVar.totalAmount - localVar.totalNonDepositAmount
        );
        uint256 compoundFee = localVar
            .totalAmount
            .percentDiv(PercentageMath.PERCENTAGE_FACTOR - localVar.compoundFee)
            .percentMul(localVar.compoundFee);
        if (compoundFee > 0) {
            APE_COMPOUND.deposit(msg.sender, compoundFee);
        }
```

When a user calls claimPairedApeAndCompound, it is required that BAYC/MAYC and BAKC have the same owner.

```
                require(
                    users[i] ==
                        INToken(localVar.xTokenAddress).ownerOf(
                            _nftPairs[i][j].mainTokenId
                        ),
                    Errors.NOT_THE_OWNER
                );

                localVar.transferredTokenOwners[
                        j
                    ] = _validateBAKCOwnerAndTransfer(
                    localVar,
                    _nftPairs[i][j].bakcTokenId,
                    users[i]    // @audit : same owner
                );
```

Since the contract currently allows users to stake using BAKC (not nBAKC), If the PairedApe staker sends BAKC to another address or cancels approval of the Pool, then other users will not be able to call claimPairedApeAndCompound to earn the compoundFee. And the PairedApe staker can transfer the BAKC back or approve the Pool and call claimPairedApeAndCompound in one transaction to earn compoundFee.

Recommendation

thereksfour : Consider only allowing users to stake with nBAKC (not BAKC)

Client Response

We call claimPairedApeAndCompound to compound for user not to earn compound fee. So if user transferred this BAKC or cancelled approval for the pool, we will not call this function for user. We'll only support nBACK for pair staking later.

PSV-4:Potential Reentrancy risk in `PoolApeStaking` contract `repayAndSupply` function

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/protocol/pool/PoolApeStaking.sol#L381**	**Declined**	**w2ning**

Code

```
381:    function repayAndSupply(
```

Description

w2ning : Missing reentrancy guard in `repayAndSupply` function.

Recommendation

w2ning : Consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Consider below fix in the `PoolApeStaking.repayAndSupply()` function

```
    function repayAndSupply(
        address underlyingAsset,
        address onBehalfOf,
        uint256 totalAmount
    ) external
      nonReentrant
    {
      ...
    }
```

Client Response

repayAndSupply can only be called by the NToken contract. And NToken contract have reentrancy guard, so this function don't have reentrancy issue.

PSV-5: `ApeStakingLogic::getTokenIdStakingAmount` should consider the BAKC reward APE token

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/protocol/tokenization/libraries/ApeStakingogic.sol#L250-L280	Acknowledged	comcat

Code

```
    function getTokenIdStakingAmount(
        uint256 poolId,
        ApeCoinStaking _apeCoinStaking,
        uint256 tokenId
    ) public view returns (uint256) {
        (uint256 apeStakedAmount, ) = _apeCoinStaking.nftPosition(
            poolId,
            tokenId
        );

        uint256 apeReward = _apeCoinStaking.pendingRewards(
            poolId,
            address(this),
            tokenId
        );

        (uint256 bakcTokenId, bool isPaired) = _apeCoinStaking.mainToBakc(
            poolId,
            tokenId
        );

        if (isPaired) {
            (uint256 bakcStakedAmount, ) = _apeCoinStaking.nftPosition(
                BAKC_POOL_ID,
                bakcTokenId
            );
            apeStakedAmount += bakcStakedAmount;
        }

        return apeStakedAmount + apeReward;
    }
```

Description

comcat : inside the `ApeStakingLogic` contract, the function `getTokenIdStakingAmount` calculates the all the ape token staking amount, including the reward amount. basically, it use the following formula to do the calculation: $$ ApeAmount = BAYCStakingAmount + BAYCRewardAmount + BAKCStakingAmount $$ however, according to the `APE staking contract`, inside the `_withdrawPairNft`, the BAKC rewarded APE token will be rewarded to the BAKC owner.

```
function _withdrawPairNft(uint256 mainTypePoolId, PairNftWithdrawWithAmount[] calldata _nfts)
private {
        ...
        for(uint256 i; i < length;) {
            ...
            if (pair.isUncommit) {
                uint256 rewardsToBeClaimed = _claim(BAKC_POOL_ID, position, bakcOwner);
                mainToBakc[mainTypePoolId][pair.mainTokenId] = PairingStatus(0, false);
                bakcToMain[pair.bakcTokenId][mainTypePoolId] = PairingStatus(0, false);
                emit ClaimRewardsPairNft(msg.sender, rewardsToBeClaimed, mainTypePoolId,
pair.mainTokenId, pair.bakcTokenId);
            }
            ...
        }
    }
```

which means that: when calculate the total APE token staking plus reward, should take the BAKC reward into consideration too. namely: $$ ApeAmount = BAYCStakingAmount + BAYCRewardAmount + BAKCStakingAmount + BAKCRewardAmount $$

Recommendation

comcat : Add the corresponding BAKC reward amount, you may refer to the following code:

```
function getTokenIdStakingAmount(
        uint256 poolId,
        ApeCoinStaking _apeCoinStaking,
        uint256 tokenId,
        address NTokenBAKC
    ) public view returns (uint256) {
        (uint256 apeStakedAmount,) = _apeCoinStaking.nftPosition(poolId, tokenId);

        uint256 apeReward = _apeCoinStaking.pendingRewards(poolId, address(this), tokenId);

        (uint256 bakcTokenId, bool isPaired) = _apeCoinStaking.mainToBakc(poolId, tokenId);

        if (isPaired) {
            (uint256 bakcStakedAmount,) =
                _apeCoinStaking.nftPosition(BAKC_POOL_ID, bakcTokenId);
            apeStakedAmount += bakcStakedAmount;
            if (bakcStakedAmount > 0) {
                bool sameOwner = INToken(NTokenBAKC).ownerOf(bakcTokenId) ==
INToken(address(this)).ownerOf(tokenId);
                if (sameOwner) {
                    apeReward += _apeCoinStaking.pendingRewards(BAKC_POOL_ID, address(NTokenBAKC),
bakcTokenId);
                }
            }
        }

        return apeStakedAmount + apeReward;
    }
```

Client Response

We didn't consider the BAKC reward ape coin as sApe balance because the BAKC reward ape coin belongs to BAKC owner. When We only support nBACK for pair staking, we can follow your advice.

PSV-6: `PoolApeStaking::_validateBAKCOwnerAndTransfer` Gas optimization

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/protocol/pool/PoolApeStaking.sol#L765-L781	Declined	comcat

Code

```
765:    function _validateBAKCOwnerAndTransfer(
766:        ApeStakingLocalVars memory localVar,
767:        uint256 tokenId,
768:        address userAddress
769:    ) internal returns (address bakcOwner) {
770:        bakcOwner = localVar.bakcContract.ownerOf(tokenId);
771:        require(
772:            (userAddress == bakcOwner) ||
773:                (userAddress == INToken(localVar.bakcNToken).ownerOf(tokenId)),
774:            Errors.NOT_THE_BAKC_OWNER
775:        );
776:        localVar.bakcContract.safeTransferFrom(
777:            bakcOwner,
778:            localVar.xTokenAddress,
779:            tokenId
780:        );
781:    }
```

Description

comcat : inside the PoolApeStaking contract, for the function `_validateBAKCOwnerAndTransfer`, it will validate the userAddress and backOwner. it first check the userAddress is bakcOwner, or the userAddress is the owner of corresponding bakcNtoken owner. after that it call the `safeTransferFrom`. however, it doesn't consider one situation, which is that the bakcOwner is the bakcNtoken contract address itself.

when i checked the BAKC contract, it is ok to transfer BAKC to the owner itself, so the call will success without problem.

```
localVar.bakcContract.safeTransferFrom(
            bakcOwner,
            localVar.xTokenAddress,
            tokenId
        );
```

but in order to save gas, or make the logic more consistent, it is better to have a check, to avoid transfer BAKC to yourself.

Recommendation

comcat : add an check, to avoid transfer BAKC to yourself.

```
function _validateBAKCOwnerAndTransfer(
        ApeStakingLocalVars memory localVar,
        uint256 tokenId,
        address userAddress
    ) internal returns (address bakcOwner) {
        ...
        if (bakcOwner == localVar.xTokenAddress) return bakcOwner;

        localVar.bakcContract.safeTransferFrom(
            bakcOwner,
            localVar.xTokenAddress,
            tokenId
        );
    }
```

Client Response

_validateBAKCOwnerAndTransfer is trying to transfer BAKC from nBAKC or user wallet to nBAYC/nMAYC. If the bakcOwner is the bakcNtoken contract address we still need to transfer it to nBAYC/nMAYC to withdraw/claim ape position

Disclaimer