



# # Competitive Security Assessment

ParaSpace V1.4 P2

Feb 17th, 2023

---

Summary	3
Overview	4
Audit Scope	5
Code Assessment Findings	6
PSV-1:Missing repeatability check for <code>tokenIds</code> & <code>_nftPairs</code> in <code>PoolApeStaking</code> contract <code>claimBAKC</code> & <code>withdrawBAKC</code> & <code>claimPairedApeAndCompound</code> function	8
PSV-2:New BAKC owners can lock the staked APECoin of BAYC/MAYC owners in the contract	10
PSV-3:PairedApe stakers can prevent other users from calling <code>claimPairedApeAndCompound</code> to earn <code>compoundFee</code>	13
PSV-4:Potential Reentrancy risk in <code>PoolApeStaking</code> contract <code>repayAndSupply</code> function	17
PSV-5: <code>ApeStakingLogic::getTokenIdStakingAmount</code> should consider the BAKC reward APE token	18
PSV-6: <code>PoolApeStaking::_validateBAKCOwnerAndTransfer</code> Gas optimization	22
Disclaimer	24

# 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	<ul style="list-style-type: none"><li>• <a href="https://github.com/para-space/paraspace-core">https://github.com/para-space/paraspace-core</a></li><li>• audit commit - 6ba3c430a24b9781fee5f0c0745862748655b36e</li><li>• final commit - 6ba3c430a24b9781fee5f0c0745862748655b36e</li></ul>
Audit Methodology	<ul style="list-style-type: none"><li>• Audit Contest</li><li>• Business Logic and Code Review</li><li>• Privileged Roles Review</li><li>• Static Analysis</li></ul>

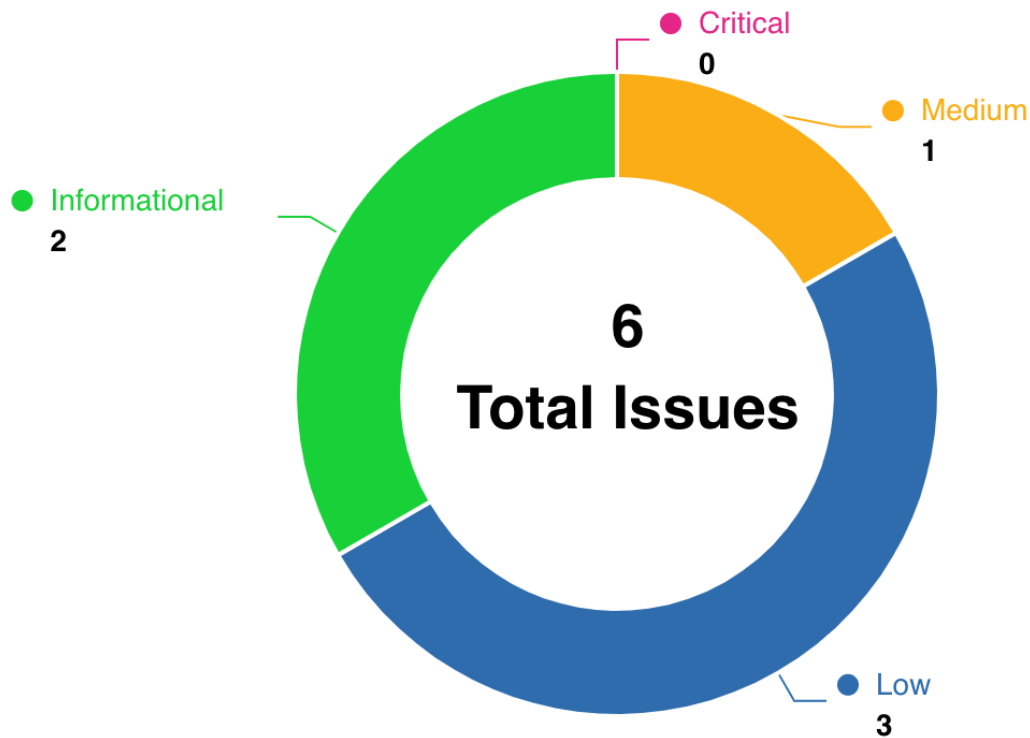
## Code Vulnerability Review Summary

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	<ul style="list-style-type: none"> <li><code>code/contracts/protocol/pool/PoolApeStaking.sol#L139</code></li> <li><code>code/contracts/protocol/pool/PoolApeStaking.sol#L193</code></li> <li><code>code/contracts/protocol/pool/PoolApeStaking.sol#L446</code></li> </ul>	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	<ul style="list-style-type: none"><li>code/contracts/protocol/pool/PoolApeStaking.sol#L765-L781</li></ul>	Acknowledged	thereksfour

### 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

**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	<ul style="list-style-type: none"><li>code/contracts/protocol/pool/PoolApeStaking.sol#L446-L502</li></ul>	Acknowledged	thereksfour

### Code

```
446:     function claimPairedApeAndCompound(  
447:         address nftAsset,  
448:         address[] calldata users,  
449:         ApeCoinStaking.PairNft[][] calldata _nftPairs  
450:     ) external nonReentrant {  
451:         require(  
452:             users.length == _nftPairs.length,  
453:             Errors.INCONSISTENT_PARAMS_LENGTH  
454:         );  
455:         DataTypes.PoolStorage storage ps = poolStorage();  
456:  
457:         ApeStakingLocalVars memory localVar = _compoundCache(  
458:             ps,  
459:             nftAsset,  
460:             users.length  
461:         );  
462:  
463:         for (uint256 i = 0; i < _nftPairs.length; i++) {  
464:             localVar.transferredTokenOwners = new address[](   
465:                 _nftPairs[i].length  
466:             );  
467:             for (uint256 j = 0; j < _nftPairs[i].length; j++) {  
468:                 require(  
469:                     users[i] ==  
470:                     INToken(localVar.xTokenAddress).ownerOf(  
471:                         _nftPairs[i][j].mainTokenId  
472:                     ),  
473:                     Errors.NOT_THE_OWNER  
474:                 );  
475:  
476:                 localVar.transferredTokenOwners[  
477:                     j  
478:                 ] = _validateBAKCOwnerAndTransfer(  
479:                     localVar,  
480:                     _nftPairs[i][j].bakcTokenId,  
481:                     users[i]  
482:                 );  
483:             }  
484:  
485:             INTokenApeStaking(localVar.xTokenAddress).claimBAKC(  
486:                 _nftPairs[i],  
487:                 address(this)
```

```
488:         );
489:
490:         for (uint256 index = 0; index < _nftPairs[i].length; index++) {
491:             localVar.bakcContract.safeTransferFrom(
492:                 localVar.xTokenAddress,
493:                 localVar.transferredTokenOwners[index],
494:                 _nftPairs[i][index].bakcTokenId
495:             );
496:         }
497:
498:         _addUserToCompoundCache(ps, localVar, i, users[i]);
499:     }
500:
501:     _compoundForUsers(ps, localVar, users);
502: }
```

## 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	<ul style="list-style-type: none"><li>code/contracts/protocol/pool/PoolApeStaking.sol#L381</li></ul>	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	<ul style="list-style-type: none"><li>code/contracts/protocol/tokenization/libraries/ApeStakingLogic.sol#L250-L280</li></ul>	Acknowledged	comcat

### Code

```
250:     function getTokenIdStakingAmount(
251:         uint256 poolId,
252:         ApeCoinStaking _apeCoinStaking,
253:         uint256 tokenId
254:     ) public view returns (uint256) {
255:         (uint256 apeStakedAmount, ) = _apeCoinStaking.nftPosition(
256:             poolId,
257:             tokenId
258:         );
259:
260:         uint256 apeReward = _apeCoinStaking.pendingRewards(
261:             poolId,
262:             address(this),
263:             tokenId
264:         );
265:
266:         (uint256 bakcTokenId, bool isPaired) = _apeCoinStaking.mainToBakc(
267:             poolId,
268:             tokenId
269:         );
270:
271:         if (isPaired) {
272:             (uint256 bakcStakedAmount, ) = _apeCoinStaking.nftPosition(
273:                 BAKC_POOL_ID,
274:                 bakcTokenId
275:             );
276:             apeStakedAmount += bakcStakedAmount;
277:         }
278:
279:         return apeStakedAmount + apeReward;
280:     }
```

## 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:  $\text{ApeAmount} = \text{BAYCStakingAmount} + \text{BAYCRewardAmount} + \text{BAKCStakingAmount}$  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:  $\$ \$ \text{ApeAmount} = \text{BAYCStakingAmount} + \text{BAYCRewardAmount} + \text{BAKCStakingAmount} + \text{BAKCRewardAmount} \$ \$$

## 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	<ul style="list-style-type: none"><li>code/contracts/protocol/pool/PoolApeStaking.sol#L765-L781</li></ul>	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

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.