Competitive Security Assessment

ParaSpace - Ape Yield

Dec 27th, 2022

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 - Ape Yield
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - b698c7d3a26311bdecf519dcc83147286742ba05 • final commit - 22e094404dcbae88f8dbab9b6432627bf1576dfc
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

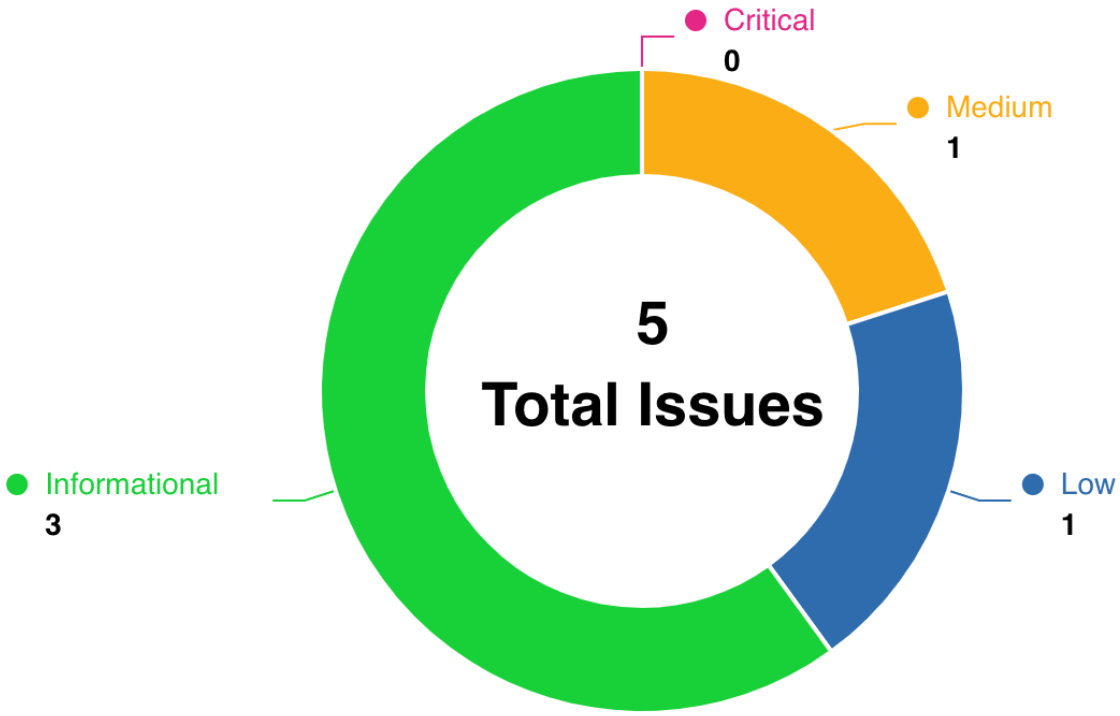
Code Vulnerability Review Summary

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

Audit Scope

File	Commit Hash
contracts/interfaces/IAutoCompoundApe.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/interfaces/ICApe.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/interfaces/IPoolApeStaking.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/interfaces/IPoolParameters.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/misc/AutoCompoundApe.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/misc/CApe.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/protocol/libraries/types/DataTypes.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/protocol/pool/PoolApeStaking.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/protocol/pool/PoolParameters.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/protocol/tokenization/CApeDebtToken.sol	b698c7d3a26311bdecf519dcc83147286742ba05
contracts/protocol/tokenization/PTokenCApe.sol	b698c7d3a26311bdecf519dcc83147286742ba05

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PSA-1	In the Pause state, users cannot redeem ApeCoin	Logical	Informational	Declined	thereksfour
PSA-2	SafeMath is unnecessary after solidity `0.8.0`	Gas Optimization	Informational	Acknowledged	comcat, jayphbee, Hupixiong3
PSA-3	User can `withdraw` for free.	Logical	Medium	Acknowledged	jayphbee
PSA-4	`_transfer` duplicate checks for the zero address	Gas Optimization	Informational	Fixed	Hupixiong3

PSA-5	**claimApeAndCompound() may put the user under the liquidation threshold**	**Logical**	**Low**	**Acknowledged**	**thereksfour**

PSA-1:In the Pause state, users cannot redeem ApeCoin

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/misc/AutoCompoundApe.sol#L48-L64 • code/contracts/misc/CApe.sol#L391-L404	Declined	thereksfour

Code

```
48:    function withdraw(uint256 amount) external override {
49:        require(amount > 0, "zero amount");
50:
51:        uint256 amountShare = getShareByPooledApe(amount);
52:        _burn(msg.sender, amountShare);
53:
54:        _harvest();
55:        uint256 _bufferBalance = bufferBalance;
56:        if (amount > _bufferBalance) {
57:            _withdrawFromApeCoinStaking(amount - _bufferBalance);
58:        }
59:        _transferTokenOut(msg.sender, amount);
60:
61:        _compound();
62:
63:        emit Redeem(msg.sender, amount, amountShare);
64:    }

391:    function _burn(address account, uint256 sharesAmount)
392:        internal
393:        virtual
394:        whenNotPaused
395:    {
396:        require(account != address(0), "burn from the zero address");
397:
398:        shares[account] = shares[account].sub(
399:            sharesAmount,
400:            "burn amount exceeds balance"
401:        );
402:        _totalShare = _totalShare.sub(sharesAmount);
403:        emit Transfer(account, address(0), getPooledApeByShares(sharesAmount));
404:    }
```

Description

thereksfour : When the user calls AutoCompoundApe.withdraw to redeem ApeCoin, CApe._burn will be called.

```
    function withdraw(uint256 amount) external override {
        require(amount > 0, "zero amount");

        uint256 amountShare = getShareByPooledApe(amount);
        _burn(msg.sender, amountShare);
```

However, CApe._burn can only be called in the non-Pause state, which means that if the system is in the Pause state for some reason, the user's deposited ApeCoin will not be redeemed.

```
    function _burn(address account, uint256 sharesAmount)
        internal
        virtual
        whenNotPaused
    {
```

This is not necessary, users should be able to redeem their ApeCoin at any time, and the design is likely to affect the reputation of the project.

Recommendation

thereksfour : Removing the whenNotPaused modifier from CApe._burn

```
    function _burn(address account, uint256 sharesAmount)
        internal
        virtual
-       whenNotPaused
    {
```

Client Response

It's our product design. We did this to prevent extreme problems.

PSA-2:SafeMath is unnecessary after solidity `0.8.0`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/misc/AutoCompoundApe.sol#L13 • code/contracts/misc/AutoCompoundApe.sol#13 • code/contracts/misc/CApe.sol#L17 • code/contracts/misc/CApe.sol#17	Acknowledged	comcat, jayphbee, Hupixiong3

Code

```
13:    using SafeMath for uint256;

13:    using SafeMath for uint256;

17:    using SafeMath for uint256;

17:    using SafeMath for uint256;
```

Description

comcat : in the solidity 0.8.10, the compiler handles the basic math operation well, if it overflow/subflow, it will revert. so there is no need to use safeMath instead.
jayphbee : After solidity 0.8.0, SafeMath library is unnecessary.
Hupixiong3 : The compiler version specified is 0.8.0 or higher, and overflow check is built in.The SafeMath library may not be used.Use unchecked to ignore partially calculated overflow checking.This will reduce the consumption of gas.

Recommendation

comcat : remove the usage of safe math.
jayphbee : Remove SafeMath library.
Hupixiong3 : Do not use SafeMath library.

Client Response

Since SafeMath library for solidity 0.8.10 used unchecked block for it's implementation. it's almost same gas comsumtion for using SafeMath library or not. Using SafeMath library just lost some opportunities to use unchecked block directly, but we think it's ok.

PSA-3:User can `withdraw` for free.

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/misc/AutoCompoundApe.sol#L34 • code/contracts/misc/AutoCompoundApe.sol#L51	Acknowledged	jayphbee

Code

```
34:         uint256 amountShare = getShareByPooledApe(amount);

51:         uint256 amountShare = getShareByPooledApe(amount);
```

Description

jayphbee : In the `AutoCompoundApe.sol::withdraw` function, user withraw `amount` of apeCoin by burn `amountShare` calculated by `getShareByPooledApe`. The `getShareByPooledApe` function implemented like this:

```
function getShareByPooledApe(uint256 amount) public view returns (uint256) {
        uint256 totalPooledApe = _getTotalPooledApeBalance();
        if (totalPooledApe == 0) {
            return 0;
        } else {
            return (amount * _getTotalShares()) / totalPooledApe;
        }
    }
```

Here `(amount * _getTotalShares()) / totalPooledApe` could return 0 if `(amount * _getTotalShares()) < totalPooledApe`. In the `withraw` function, `amountShare` is not checked if it is 0:

```
uint256 amountShare = getShareByPooledApe(amount);
_burn(msg.sender, amountShare);
```

And then `amount` of apeCoin is transferred to `msg.sender`.

```
_transferTokenOut(msg.sender, amount);
```

That is to say, user burns 0 share but gets `amount` of apeCoin transferred out for free. Note: this also applies to the `deposit` function where user could get more shares if `getShareByPooledApe` returns 0.

Recommendation

jayphbee : Only when `amountShare > 0` burned, can apeCoin be transferred out.

```
uint256 amountShare = getShareByPooledApe(amount);
require(amountShare > 0, "whatever");
 _burn(msg.sender, amountShare);
```

Client Response

It's a calculation precision issue. but since the attacker need to pay transaction fee for it, and the transaction fee value is always much greater than the amount of ApeCoin value they get. The attacker has no economic incentive to do so, so we think it's ok.

PSA-4: `_transfer` duplicate checks for the zero address

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/misc/CApe.sol#L344-L357	Fixed	Hupixiong3

Code

```
344:    function _transferShares(
345:        address _sender,
346:        address _recipient,
347:        uint256 _sharesAmount
348:    ) internal whenNotPaused {
349:        require(_sender != address(0), "TRANSFER_FROM_THE_ZERO_ADDRESS");
350:        require(_recipient != address(0), "TRANSFER_TO_THE_ZERO_ADDRESS");
351:
352:        shares[_sender] = shares[_sender].sub(
353:            _sharesAmount,
354:            "transfer amount exceeds balance"
355:        );
356:        shares[_recipient] = shares[_recipient].add(_sharesAmount);
357:    }
```

Description

Hupixiong3 : The `_transferShares` function is used only for `_transfer` function, and the `_transfer` function has already performed 0 address check.

Recommendation

Hupixiong3 : Cancel the _transferShares function or 0 address check.

Consider below fix in the `_transferShares` function

```
    function _transferShares(
        address _sender,
        address _recipient,
        uint256 _sharesAmount
    ) internal whenNotPaused {
        shares[_sender] = shares[_sender].sub(
            _sharesAmount,
            "transfer amount exceeds balance"
        );
        shares[_recipient] = shares[_recipient].add(_sharesAmount);
    }
```

Or Consider below fix in the `_transfer` function

```
    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual whenNotPaused{
        require(sender != address(0), "transfer from the zero address");
        require(recipient != address(0), "transfer to the zero address");

        uint256 _sharesToTransfer = getShareByPooledApe(amount);
        shares[_sender] = shares[_sender].sub(
            _sharesAmount,
            "transfer amount exceeds balance"
        );
        shares[_recipient] = shares[_recipient].add(_sharesAmount);
        emit Transfer(sender, recipient, amount);
    }
```

Client Response

fixed

PSA-5:claimApeAndCompound() may put the user under the liquidation threshold

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/pool/PoolApeStaking.sol#L436-L490	Acknowledged	thereksfour

Code

```
436:     function claimApeAndCompound(
437:         address nftAsset,
438:         address[] calldata users,
439:         uint256[][] calldata tokenIds
440:     ) external nonReentrant {
441:         require(users.length == tokenIds.length, "invalid parameter");
442:         DataTypes.PoolStorage storage ps = poolStorage();
443:         checkSApeIsNotPaused(ps);
444: 
445:         address xTokenAddress = ps._reserves[nftAsset].xTokenAddress;
446: 
447:         uint256 balanceBefore = APE_COIN.balanceOf(address(this));
448:         uint256[] memory amounts = new uint256[](tokenIds.length);
449: 
450:         uint256 totalAmount;
451:         for (uint256 i = 0; i < tokenIds.length; i++) {
452:             uint256[] calldata userTokenIds = tokenIds[i];
453:             for (uint256 j = 0; j < userTokenIds.length; j++) {
454:                 address positionOwner = INToken(xTokenAddress).ownerOf(
455:                     userTokenIds[j]
456:                 );
457:                 require(users[i] == positionOwner, "user is not owner");
458:             }
459: 
460:             INTokenApeStaking(xTokenAddress).claimApeCoin(
461:                 userTokenIds,
462:                 address(this)
463:             );
464: 
465:             uint256 balanceAfter = APE_COIN.balanceOf(address(this));
466:             amounts[i] = balanceAfter - balanceBefore;
467:             balanceBefore = balanceAfter;
468:             totalAmount += amounts[i];
469:         }
470: 
471:         uint256 compoundFee = ps._apeCompoundFee;
472:         uint256 totalFee = totalAmount.percentMul(compoundFee);
473:         APE_COMPOUND.deposit(address(this), totalAmount);
474: 
475:         if (totalFee > 0) {
476:             IERC20(address(APE_COMPOUND)).safeTransfer(msg.sender, totalFee);
477:         }
```

```
478:
479:         for (uint256 index = 0; index < users.length; index++) {
480:             if (amounts[index] != 0) {
481:                 _supplyCApeForUser(
482:                     ps,
483:                     users[index],
484:                     amounts[index].percentMul(
485:                         PercentageMath.PERCENTAGE_FACTOR - compoundFee
486:                     )
487:                 );
488:             }
489:         }
490:     }
```

Description

thereksfour : Anyone can call PoolApeStaking.claimApeAndCompound to deposit any user's unclaimed ApeCoin rewards in ApeCoinStaking into APE_COMPOUND to be converted to CApe, where a portion of the CApe is sent to the caller as an incentive.

```
            INTokenApeStaking(xTokenAddress).claimApeCoin(
                userTokenIds,
                address(this)
            );

            uint256 balanceAfter = APE_COIN.balanceOf(address(this));
            amounts[i] = balanceAfter - balanceBefore;
            balanceBefore = balanceAfter;
            totalAmount += amounts[i];
        }

        uint256 compoundFee = ps._apeCompoundFee;
        uint256 totalFee = totalAmount.percentMul(compoundFee);
        APE_COMPOUND.deposit(address(this), totalAmount);

        if (totalFee > 0) {
            IERC20(address(APE_COMPOUND)).safeTransfer(msg.sender, totalFee);
        }
```

And _apeCompoundFee can be up to 50%

```
    function setClaimApeForCompoundFee(uint256 fee) external onlyPoolAdmin {
        require(fee < PercentageMath.HALF_PERCENTAGE_FACTOR, "Value Too High");
        DataTypes.PoolStorage storage ps = poolStorage();
        uint256 oldValue = ps._apeCompoundFee;
        if (oldValue != fee) {
            ps._apeCompoundFee = uint16(fee);
            emit ClaimApeForYieldIncentiveUpdated(oldValue, fee);
        }
    }
```

Since unclaimed ApeCoin rewards in ApeCoinStaking are also part of the user's collateral, claimApeAndCompound actually reduces the user's collateral and is likely to bring the user below the liquidation threshold, putting the user at risk of being liquidated.

Recommendation

thereksfour : Add Health Factor checks for users in PoolApeStaking.claimApeAndCompound to avoid bringing users below the liquidation threshold

Client Response

This is indeed a problem. but it'll cost much more gas if we perform a health factor check. We choose to keep it like this and have a very low _apeCompoundFee value to reduce the potential risk.

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.