Competitive Security Assessment

ParaSpace yAPE

Mar 23rd, 2023

Secure3

secure3.io

Summary 3

Overview 4

Audit Scope 5

Code Assessment Findings 6

PSY-1:ApeCoin token address are marked as immutable 8

PSY-2:Miss 0 amount check for `autocompoundape::withdraw()` 9

PSY-3:The `withdrawFee` in `PYieldToken` will be locked in the contract 11

PSY-4:The security library is not used correctly 13

PSY-5:Using OpenZeppelin's libraries with vulnerabilities 15

PSY-6:Using `deprecated` function from library 16

PSY-7:Without `from!=to` check in `PYieldToken::_transfer::withdrawFee` 19

PSY-8:`Pyieldtoken::_updateUserIndex()`: `WithdrawLockAmount` without pay `withdrawFee` 23

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 yAPE
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - 7bb3e5151197eb57a6875238ffeba26fb7f069c8 • final commit - f4191290147a62c99ad83133908cfa576a50e0d6
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

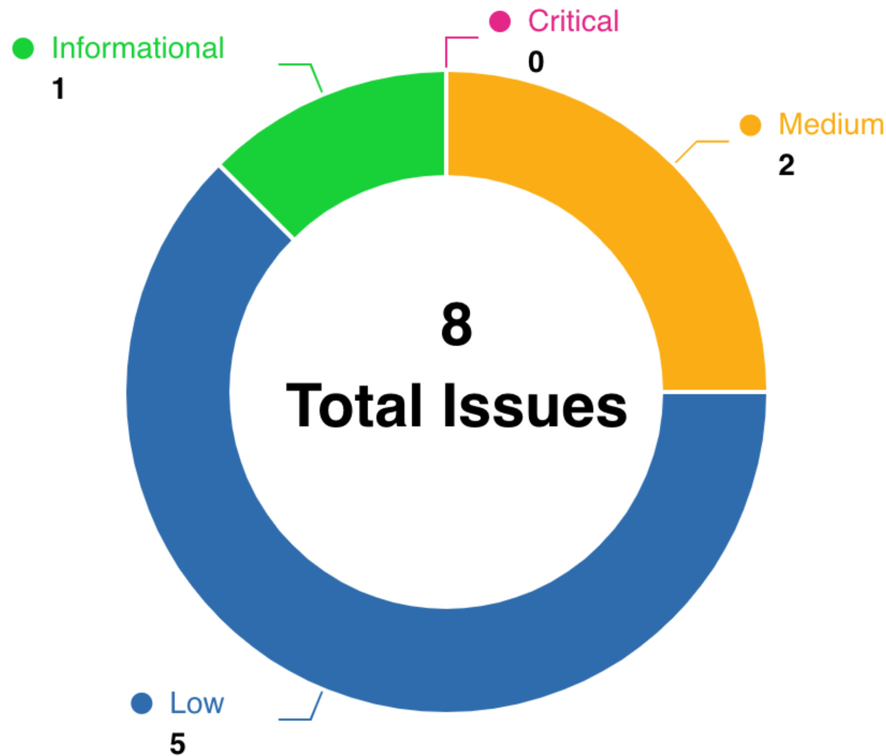
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	2	0	1	1	0	0
Low	5	0	3	1	0	1
Informational	1	0	1	0	0	0

Audit Scope

File	Commit Hash
contracts/misc/AutoCompoundApe.sol	7bb3e5151197eb57a6875238ffeba26fb7f069c8
contracts/misc/AutoYieldApe.sol	7bb3e5151197eb57a6875238ffeba26fb7f069c8
contracts/misc/VoteDelegator.sol	7bb3e5151197eb57a6875238ffeba26fb7f069c8
contracts/protocol/tokenization/PToken.sol	7bb3e5151197eb57a6875238ffeba26fb7f069c8
contracts/protocol/tokenization/PYieldToken.sol	7bb3e5151197eb57a6875238ffeba26fb7f069c8
contracts/protocol/tokenization/VariableDebtToken.sol	7bb3e5151197eb57a6875238ffeba26fb7f069c8

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PSY-1	ApeCoin token address are marked as immutable	DOS	Informational	Acknowledged	0xzoobi
PSY-2	Miss 0 amount check for `autocompoundape::withdraw()`	Logical	Low	Acknowledged	8olidity
PSY-3	The `withdrawFee` in `PYieldToken` will be locked in the contract	Logical	Medium	Acknowledged	thereksfour
PSY-4	The security library is not used correctly	Code Style	Low	Fixed	8olidity

PSY-5	Using OpenZeppelin's libraries with vulnerabilities	Code Style	Low	Declined	8olidity
PSY-6	Using `deprecated` function from library	Code Style	Low	Acknowledged	8olidity
PSY-7	Without `from!=to` check in `PYieldToken::_transfer::withdrawFee`	Logical	Medium	Fixed	thereksfour, 8olidity
PSY-8	`Pyieldtoken::_updateUserIndex()` : `WithdrawLockAmount` without pay `withdrawFee`	Logical	Low	Acknowledged	8olidity

PSY-1:ApeCoin token address are marked as immutable

Category	Severity	Code Reference	Status	Contributor
DOS	Informational	• code/contracts/misc/AutoCompoundApe.sol#L30 • code/contracts/misc/AutoYieldApe.sol#L37	Acknowledged	0xzoobi

Code

```
30:    IERC20 public immutable apeCoin;

37:    address private immutable _apeCoin;
```

Description

0xzoobi : The whole project is based around Optimizing and giving maximum benefit to users who want to stake their ApeCoin ERC20 tokens. There can be scenario in the future, where in Yuga Labs can migrate the token to a V2. The contracts are non upgradable, hence they need to migrate to an new address, this could be to fix a vulnerability or introduce new features.

The erc20 tokens have been declared as `immutable`, which means they cannot be updated on the code once deployed.

There are two scenarios that may happen in the future.

1. Total migration of ApecoinV1 and ApecoinV2, which means the deployment takes place on a new address. This can result in the ApecoinV1 useless.
2. Both V1 and V2 might co-exist like Uniswap but this is less likely to happen. In this case, Paraspace wont be able to access the latest features of V2.

Recommendation

0xzoobi : A better approach would be to add an `external` function with `onlyOwner` access to modify or update the ApeCoin token address.

Client Response

Our yApe will be deployed in an upgradable way. So it’s not an issue.

PSY-2:Miss 0 amount check for `autocompoundape::withdraw()`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/misc/AutoCompoundApe.sol#L65-L82	Acknowledged	8olidity

Code

```
65:     function withdraw(uint256 amount) external override {
66:         require(amount > 0, "zero amount");
67:
68:         uint256 amountShare = getShareByPooledApe(amount);
69:         _burn(msg.sender, amountShare);
70:
71:         _harvest();
72:         uint256 _bufferBalance = bufferBalance;
73:         if (amount > _bufferBalance) {
74:             _withdrawFromApeCoinStaking(amount - _bufferBalance);
75:         }
76:         _transferTokenOut(msg.sender, amount);
77:
78:         _compound();
79:
80:         emit Transfer(msg.sender, address(0), amount);
81:         emit Redeem(msg.sender, amount, amountShare);
82:     }
```

Description

8olidity : In `autocompoundape::withdraw()`, the tokens of the user's `amountShare` will be burned,

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

and this value is calculated by `getShareByPooledApe()`,Due to solidity rounding, this value may be 0

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

But the contract will still send the user the amount of tokens

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

Recommendation

8olidity : Exit function when amountshare = 0

Client Response

For cApe withdraw, this can happen for any input amount. This is due to precision loss and can not be solved. It’s meaningless just checking 0 amounts.

PSY-3:The `withdrawFee` in `PYieldToken` will be locked in the contract

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/tokenization/PYieldToken.sol#L165-L171 • code/contracts/misc/AutoYieldApe.sol#L370-L382	Acknowledged	thereksfour

Code

```
165:            if (leftBalance < userLockFeeBalance) {
166:                uint256 withdrawLockAmount = userLockFeeBalance - leftBalance;
167:                uint256 withdrawFee = (withdrawLockAmount * lastAccruedIndex) /
168:                    RAY;
169:                _userLockFeeAmount[account] -= withdrawLockAmount;
170:                _userPendingYield[account] -= withdrawFee;
171:            }

370:        if (balanceDiff < 0) {
371:            uint256 leftBalance = userBalance - (uint256(-balanceDiff));
372:            uint256 userLockFeeBalance = _userLockFeeAmount[account];
373:            //here we only need to update lock fee amount and charge fee when reduce user lock
fee amount
374:            if (leftBalance < userLockFeeBalance) {
375:                uint256 withdrawLockAmount = userLockFeeBalance - leftBalance;
376:                uint256 withdrawFee = (withdrawLockAmount *
377:                    _poolLastAccruedIndex) / RAY;
378:                _userLockFeeAmount[account] -= withdrawLockAmount;
379:                _userPendingYield[account] -= withdrawFee;
380:                _userPendingYield[owner()] += withdrawFee;
381:            }
382:        }
```

Description

thereksfour : AutoYieldApe will mint yAPE for users who deposit ApeCoin, and users can deposit yAPE into PYieldToken to mint pyAPE. When a user transfers yAPE or pyAPE, the last reward is deducted as withdrawFee. yAPE and pyAPE

use the same lastAccruedIndex and latestYieldIndex. The difference, however, is that yAPE adds withdrawFee to _userPendingYield[owner()], while pyAPE does not, which causes the withdrawFee in pyAPE to be locked in the contract. This is because when the pyAPE is transferred, the yAPE is not actually transferred and is still held in the pyAPE contract.

Consider alice deposits yAPE to mint pyAPE, and after a period of time generating a profit of 1000 pUSDC, alice transfers the pyAPE to bob and 100 pUSDC is deducted as the last profit. But since the yAPE in pyAPE has not been transferred, that is, pyAPE still has 1000 pUSDC of profit, but alice and bob can only take out 9900 pUSDC, leaving 100 pUSDC locked in the contract.

Recommendation

thereksfour : Consider adding FEE_RECIPIENT to the PYieldToken to charge the withdrawalFee.

```
+          address  FEE_RECIPIENT = 0x...;
            if (leftBalance < userLockFeeBalance) {
                uint256 withdrawLockAmount = userLockFeeBalance - leftBalance;
                uint256 withdrawFee = (withdrawLockAmount * lastAccruedIndex) /
                    RAY;
                _userLockFeeAmount[account] -= withdrawLockAmount;
                _userPendingYield[account] -= withdrawFee;
+               _userPendingYield[FEE_RECIPIENT] += withdrawFee;
            }
```

Or consider not charging withdrawFee in PYieldToken

Client Response

We intend to do it. The Withdrawal fee mechanism is just for preventing arbitrage. It will need more gas to keep it recorded. And since our PYiledToken is also deployed in an upgradable way, it will not be locked.

PSY-4:The security library is not used correctly

Category	Severity	Code Reference	Status	Contributor
Code Style	**Low**	• **code/contracts/protocol/tokenization/PYieldToken.sol#L43** • **code/contracts/protocol/tokenization/PYieldToken.sol#L54** • **code/contracts/protocol/tokenization/PYieldToken.sol#L68** • **code/contracts/protocol/tokenization/PYieldToken.sol#L69** • **code/contracts/protocol/tokenization/PYieldToken.sol#L162**	**Fixed**	**8olidity**

Code

```
43:        _updateUserIndex(onBehalfOf, int256(amount));

54:        _updateUserIndex(from, -int256(amount));

68:        _updateUserIndex(from, -int256(amount));

69:        _updateUserIndex(to, int256(amount));

162:            uint256 leftBalance = userBalance - (uint256(-balanceDiff));
```

Description

8olidity : `PYieldToken.sol` refers to SafeCast, but it is not used correctly, such as in Mint functions

```
function mint(
        address caller,
        address onBehalfOf,
        uint256 amount,
        uint256 index
    ) external override onlyPool returns (bool) {
        _updateUserIndex(onBehalfOf, int256(amount));

        return _mintScaled(caller, onBehalfOf, amount, index);
    }
```

Recommendation

8olidity : The code converts the amount directly to int256, it is safer to use `amount.toInt256()` here

Client Response

Fixed

PSY-5:Using OpenZeppelin's libraries with vulnerabilities

Category	Severity	Code Reference	Status	Contributor
Code Style	Low	• code/package.json#L33-L34	Declined	8olidity

Code

```
33:    "@openzeppelin/contracts": "4.2.0",
34:    "@openzeppelin/contracts-upgradeable": "4.2.0",
```

Description

8olidity : Using vulnerable dependency of OpenZeppelin,The package.json configuration file says that the project is using 4.2.0 of OZ which has a not last update version:

```
    "@openzeppelin/contracts": "4.2.0",
    "@openzeppelin/contracts-upgradeable": "4.2.0",
```

poc https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-4h98-2769-gh6h

Recommendation

8olidity : Use patched versions. Latest non vulnerable version 4.8.0.

Client Response

Our solidity code does not use dependency specified by package.json

PSY-6:Using `deprecated` function from library

Category	Severity	Code Reference	Status	Contributor
Code Style	Low	• code/contracts/misc/AutoYieldApe.sol#L79-L116	Acknowledged	8olidity

Code

```
79:     function initialize() public initializer {
80:         __Ownable_init();
81:         __ERC20_init("ParaSpace Auto Yield APE", "yAPE");
82:
83:         //approve ApeCoin for apeCoinStaking
84:         uint256 allowance = IERC20(_apeCoin).allowance(
85:             address(this),
86:             address(_apeStaking)
87:         );
88:         if (allowance == 0) {
89:             IERC20(_apeCoin).safeApprove(
90:                 address(_apeStaking),
91:                 type(uint256).max
92:             );
93:         }
94:         //approve _yieldUnderlying for lending pool
95:         allowance = IERC20(_yieldUnderlying).allowance(
96:             address(this),
97:             address(_lendingPool)
98:         );
99:         if (allowance == 0) {
100:             IERC20(_yieldUnderlying).safeApprove(
101:                 address(_lendingPool),
102:                 type(uint256).max
103:             );
104:         }
105:         //approve ApeCoin for uniswap
106:         allowance = IERC20(_apeCoin).allowance(
107:             address(this),
108:             address(_swapRouter)
109:         );
110:         if (allowance == 0) {
111:             IERC20(_apeCoin).safeApprove(
112:                 address(_swapRouter),
113:                 type(uint256).max
114:             );
115:         }
116:     }
```

Description

8olidity : Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()`. If only setting the initial allowance to the value that means infinite, `safeIncreaseAllowance()` can be used instead

```
    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
```

Recommendation

8olidity : As suggested by the OpenZeppelin comment, replace `safeApprove()` with `safeIncreaseAllowance()` or `safeDecreaseAllowance()` instead.

Client Response

It's a one-time operation, so it’s ok.

PSY-7:Without `from!=to` check in `PYieldToken::_transfer::withdrawFee`

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/tokenization/PYieldToken.sol#L62-L72 • code/contracts/protocol/tokenization/PYieldToken.sol#L140-L170 • code/contracts/misc/AutoYieldApe.sol#L409-L418	Fixed	thereksfour, 8olidity

Code

```
62:     function _transfer(
63:         address from,
64:         address to,
65:         uint256 amount,
66:         bool validate
67:     ) internal override {
68:         _updateUserIndex(from, -int256(amount));
69:         _updateUserIndex(to, int256(amount));
70:
71:         super._transfer(from, to, amount, validate);
72:     }

140:     function _updateUserIndex(address account, int256 balanceDiff) internal {
141:         uint256 userBalance = balanceOf(account);
142:         (uint256 lastAccruedIndex, uint256 latestYieldIndex) = IYieldInfo(
143:             _underlyingAsset
144:         ).yieldIndex();
145:         uint256 indexDiff = latestYieldIndex - _userYieldIndex[account];
146:         uint256 pendingYield = _userPendingYield[account];
147:         //update pending yield and user lock fee amount first if necessary
148:         if (indexDiff > 0) {
149:             if (userBalance > 0) {
150:                 uint256 accruedYield = (userBalance * indexDiff) / RAY;
151:                 pendingYield += accruedYield;
152:                 if (userBalance != _userLockFeeAmount[account]) {
153:                     _userLockFeeAmount[account] = userBalance;
154:                 }
155:                 _userPendingYield[account] = pendingYield;
156:             }
157:             _userYieldIndex[account] = latestYieldIndex;
158:         }
159:
160:         //if it's the withdraw or transfer balance out case
161:         if (balanceDiff < 0) {
162:             uint256 leftBalance = userBalance - (uint256(-balanceDiff));
163:             uint256 userLockFeeBalance = _userLockFeeAmount[account];
164:             //here we only need to update lock fee amount and charge fee when reduce user lock
fee amount
165:             if (leftBalance < userLockFeeBalance) {
166:                 uint256 withdrawLockAmount = userLockFeeBalance - leftBalance;
167:                 uint256 withdrawFee = (withdrawLockAmount * lastAccruedIndex) /
168:                     RAY;
```

```
169:                _userLockFeeAmount[account] -= withdrawLockAmount;
170:                _userPendingYield[account] -= withdrawFee;

409:    function _transfer(
410:        address sender,
411:        address recipient,
412:        uint256 amount
413:    ) internal override {
414:        require(sender != recipient, "same address for transfer");
415:        _updateYieldIndex(sender, -int256(amount));
416:        _updateYieldIndex(recipient, int256(amount));
417:        super._transfer(sender, recipient, amount);
418:    }
```

Description

thereksfour : AutoYieldApe._transfer requires sender ! = recipient, which avoids charging withdrawFee in _updateYieldIndex when sender==recipient.

However, in PYieldToken._transfer, there is no requirement that from ! = to, which means that when from == to, the _transfer will execute normally and charge the withdrawalFee

8olidity : `PYieLdToken.sol` does not limit itself to transfer money to itself, when the user transfers money to himself, it will call `_updateUserIndex()` to update

When `balanceDiff` is less than 0, the sender's `_userLockFeeAmount` and `_userPendingYield` will be updated. But from a macro point of view, when you transfer money to yourself, this value should not be updated.

POC

```
// add PYieldToken.sol
function getPendingYield(address account) external view returns (uint) {
    return _userPendingYield[account];
}

// code/test/auto_yield_ape.spec.ts:
// add
536      console.log(await yApePToken.getPendingYield(user2.address)); // out 1794598679
537:     console.log(await yApePToken.balanceOf(user2.address));
538:     await
yApePToken.connect(user2.signer).transfer(user2.address,yApePToken.balanceOf(user2.address));
539      console.log(await yApePToken.getPendingYield(user2.address)); // out 0
```

The user's _userPendingYield is updated

Recommendation

thereksfour : Consider requiring from != to in PYieldToken._transfer, or not calling _updateUserIndex when from == to.

```
    function _transfer(
        address from,
        address to,
        uint256 amount,
        bool validate
    ) internal override {
+         require(from != to, "same address for transfer");
        _updateUserIndex(from, -int256(amount));
        _updateUserIndex(to, int256(amount));

        super._transfer(from, to, amount, validate);
    }
```

8olidity : Limit yourself to sending tokens to yourself

Client Response

Fixed

PSY-8: `Pyieldtoken::_updateUserIndex()` : `WithdrawLockAmount` without pay `withdrawFee`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/tokenization/PYieldToken.sol#L170	Acknowledged	8olidity

Code

```
170:                _userPendingYield[account] -= withdrawFee;
```

Description

8olidity : In `Pyieldtoken::_updateUserIndex()`, when `banlanceDiff<0`, `_userLockFeeAmount` and `_userPendingYield` will be updated, where the value of `withdrawFee` is calculated as follows

```
uint256 withdrawFee = (withdrawLockAmount * lastAccruedIndex) / RAY;
```

Among them, the value of `RAY` is 1e27, which is very large. Combined with the rounding feature of solidity, the `withdrawFee` may be 0, that is to say, the attacker operates multiple times by manipulating the input value. Make `withdrawFee` is 0. bypass fees

Recommendation

8olidity : Determine whether `withdrawFee` is 0

Client Response

It’s true, but attackers need to pay a transaction fee to do so, it has no incentive from an economic perspective.

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.