Competitive Security Assessment

ParaSpace cAPE P2

Mar 26th, 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 cAPE P2
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - 684dd70b9c76c47c9742ae4bbcf4d645090c58cf • final commit - d3263e22565e7715c12a145313a615cde50a03fc
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

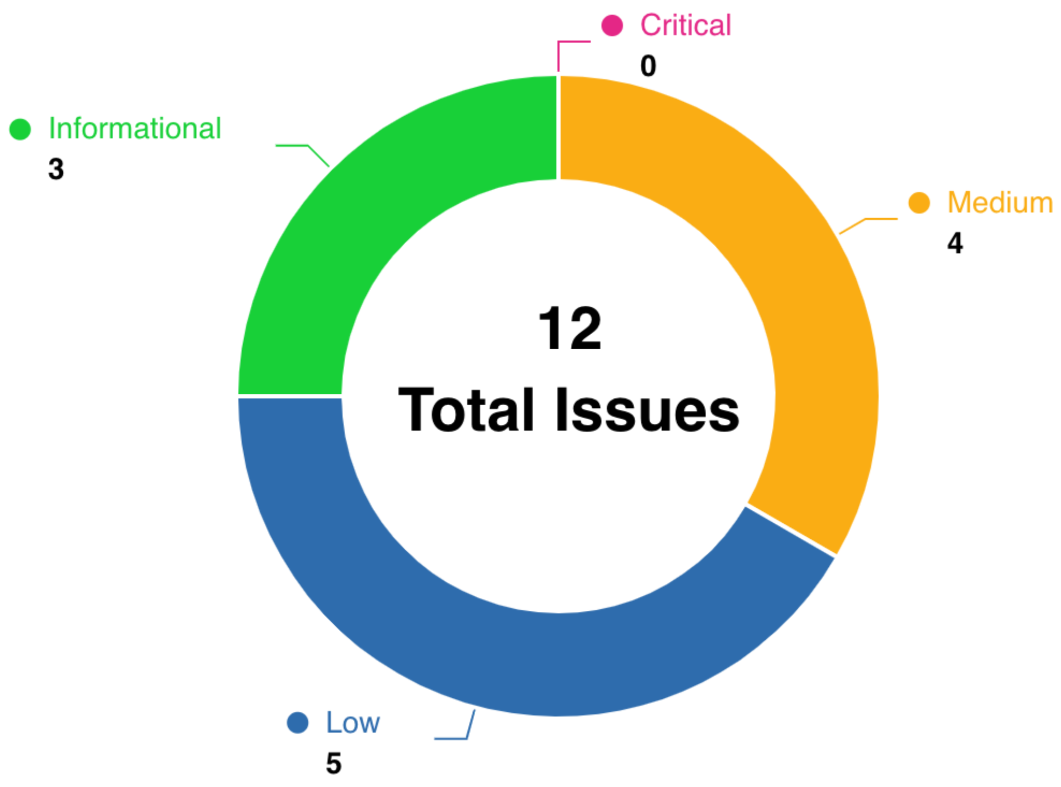
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**0**	**0**	**0**	**0**	**0**	**0**
Medium	**4**	**0**	**1**	**2**	**0**	**1**
Low	**5**	**1**	**3**	**0**	**0**	**1**
Informational	**3**	**0**	**2**	**0**	**0**	**1**

Audit Scope

File	Commit Hash
contracts/misc/AutoCompoundApe.sol	684dd70b9c76c47c9742ae4bbcf4d645090c58cf
contracts/protocol/libraries/logic/ReserveLogic.sol	684dd70b9c76c47c9742ae4bbcf4d645090c58cf
contracts/protocol/libraries/logic/SupplyLogic.sol	684dd70b9c76c47c9742ae4bbcf4d645090c58cf
contracts/protocol/libraries/logic/ValidationLogic.sol	684dd70b9c76c47c9742ae4bbcf4d645090c58cf
contracts/protocol/pool/PoolCore.sol	684dd70b9c76c47c9742ae4bbcf4d645090c58cf

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PSC-1	Additional yield does not match bufferStakingBalance	Logic	Medium	Fixed	Kong7ych3
PSC-2	Did not judge the actual payment reward	Logical	Medium	Fixed	Hupixiong3
PSC-3	Potential Reentrancy Attack	Reentrancy	Low	Acknowledged	BradMoonUESTC
PSC-4	Unchecked Return Value	Logical	Medium	Declined	BradMoonUESTC

PSC-5	**Unused return value**	**Code Style**	**Informational**	**Acknowledged**	**BradMoonUESTC**
PSC-6	**Use of unified fund withdrawal function**	**Logical**	**Low**	**Declined**	**Hupixiong3**
PSC-7	`_getTotalPooledApeBalance` **may be manipulated by** `rewardAmount`	**Logical**	**Medium**	**Acknowledged**	**Secure3**
PSC-8	`stakingBalance` **should use** `realWithdraw` **value, not use the input param** `amount`	**Code Style**	**Informational**	**Declined**	**xfu**
PSC-9	`tmp_fix_withdrawFromApeCoinStaking` **should only be allowed once**	**Logical**	**Low**	**Acknowledged**	**Secure3**
PSC-10	**event duplicate in AutoCompoundApe.sol**	**Gas Optimization**	**Informational**	**Acknowledged**	**xfu**
PSC-11	**liquidate the hacker for profit**	**Logical**	**Low**	**Acknowledged**	**comcat**
PSC-12	**tmp_fix_withdrawFromApeCoinStaking may be suffering from front-run attack**	**Logical**	**Low**	**Reported**	**thereksfour**

PSC-1:Additional yield does not match bufferStakingBalance

Category	Severity	Code Reference	Status	Contributor
Logic	**Medium**	• **code/contracts/misc/AutoCompoundApe.sol#L111**	**Fixed**	**Kong7ych3**

Code

```
111:        stakingBalance -= amount;
```

Description

Kong7ych3 : In the temporary repair function for exchange rate, a fixed amount is used for withdrawal to update bufferStakingBalance. If other users still stake for the cAPE contract before the repair is completed, the cAPE contract will receive additional benefits. This part of the income will be distributed to users, so the amount that all users can withdraw in cAPE will be larger than expected. This should be a good thing, but unfortunately bufferStakingBalance will not be updated accordingly, which will cause a user to revert in the future when performing the _withdrawFromApeCoinStaking operation because the withdrawn amount will be greater than bufferStakingBalance, resulting in user funds being locked.

```
    function _withdrawFromApeCoinStaking(uint256 amount) internal {
        ...
        bufferStakingBalance -= amount;
        ...
    }
```

Recommendation

Kong7ych3 : It is recommended to add a function to update the bufferStakingBalance parameter to avoid the above issue.

Consider below fix:

```
    function rebaseFromApeCoinStaking() external onlyOwner {
        (bufferStakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

Client Response

Fixed.

PSC-2:Did not judge the actual payment reward

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	**- code/contracts/misc/AutoCompoundApe.sol#L134-L144**	**Fixed**	**Hupixiong3**

Code

```
    function _harvest() internal {
        uint256 rewardAmount = apeStaking.pendingRewards(
            APE_COIN_POOL_ID,
            address(this),
            0
        );
        if (rewardAmount > 0) {
            apeStaking.claimSelfApeCoin();
            bufferBalance += rewardAmount;
        }
    }
```

Description

Hupixiong3 : When the reward is claimed through the _harvest() function, the contract balance change is not determined to be consistent with rewardAmount. When there is any change or malfunction to the pledge agreement of APE, bufferBalance will update error.

Recommendation

Hupixiong3 : Adding a judgment about whether the contract balance changes in line with rewardAmount when the reward is claimed through the _harvest() function can effectively prevent errors.
Consider below fix in the `AutoCompoundApe._harvest()` function

```
    function _harvest() internal {
        uint256 rewardAmount = apeStaking.pendingRewards(
            APE_COIN_POOL_ID,
            address(this),
            0
        );
        if (rewardAmount > 0) {
            uint256 balanceBefore = apeCoin.balanceOf(address(this));
            apeStaking.claimSelfApeCoin();
            uint256 balanceAfter = apeCoin.balanceOf(address(this));
            uint256 realClaim = balanceAfter - balanceBefore;
            require(rewardAmount==realClaim,"Reward error")
            bufferBalance += rewardAmount;
        }
    }
```

Client Response

Fixed.

PSC-3:Potential Reentrancy Attack

Category	Severity	Code Reference	Status	Contributor
Reentrancy	**Low**	• **code/contracts/misc/AutoCompoundApe.sol#L14**	**Acknowledged**	**BradMoonUESTC**

Code

```
14:contract AutoCompoundApe is
```

Description

BradMoonUESTC : In the contract autocompoundape, there are a large number of potential token transfers. In this case, if there is a received hook logic, the attacker may use this to carry out re-entry attacks

Recommendation

BradMoonUESTC : Use reetrancy lock

Client Response

Acknowledged.We did not fix it because we think ApeCoin don't have received hook logic.

PSC-4:Unchecked Return Value

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	• **code/contracts/misc/AutoCompoundApe.sol#L125-L145**	**Declined**	**BradMoonUESTC**

Code

```
125:    function _compound() internal {
126:        uint256 _bufferBalance = bufferBalance;
127:        if (_bufferBalance >= MIN_OPERATION_AMOUNT) {
128:            apeStaking.depositSelfApeCoin(_bufferBalance);
129:            stakingBalance += _bufferBalance;
130:            bufferBalance = 0;
131:        }
132:    }
133:
134:    function _harvest() internal {
135:        uint256 rewardAmount = apeStaking.pendingRewards(
136:            APE_COIN_POOL_ID,
137:            address(this),
138:            0
139:        );
140:        if (rewardAmount > 0) {
141:            apeStaking.claimSelfApeCoin();
142:            bufferBalance += rewardAmount;
143:        }
144:    }
145:
```

Description

BradMoonUESTC : The AutoCompoundApe contract implements an automatic compounding mechanism for the Ape token by using the ApeCoinStaking contract. The contract has functions for depositing, withdrawing, harvesting rewards, and compounding the Ape token. However, there are potential security issues with the compound and harvest functions, which may result in incorrect updates to the bufferBalance variable.

The compound function uses the depositSelfApeCoin function of the ApeCoinStaking contract to deposit Ape tokens and updates the bufferBalance variable. However, if the deposit operation fails or returns an unexpected value, the bufferBalance variable may not be updated correctly. Similarly, the harvest function uses the claimSelfApeCoin function to claim rewards and updates the bufferBalance variable, which may also be affected by unexpected returns or failures.

Recommendation

BradMoonUESTC : To address these potential security issues, we recommend adding error handling and checks in the compound and harvest functions to ensure that the deposit and claim operations are successful and return the expected values. Additionally, it is essential to monitor the ApeCoinStaking contract for any changes or potential vulnerabilities that may affect the functionality of the AutoCompoundApe contract. Furthermore, it is recommended to use the ReentrancyGuard to prevent re-entrancy attacks and to follow best practices for secure coding and contract development. Finally, it is essential to test the contract thoroughly and perform audits by security experts to ensure its safety and reliability.

Client Response

Declined. These functions don't have a return value.

PSC-5:Unused return value

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/protocol/libraries/logic/SupplyLogic.sol#L118-L122 • code/contracts/protocol/pool/PoolCore.sol#L504-L522 • code/contracts/protocol/pool/PoolCore.sol#L535-L553	Acknowledged	BradMoonUESTC

Code

```
                    IVariableDebtToken(debtTokenAddress).burn(
                        from,
                        debtBalance,
                        borrowIndex
                    );

        LiquidationLogic.executeLiquidateERC20(
            ps._reserves,
            ps._reservesList,
            ps._usersConfig,
            DataTypes.ExecuteLiquidateParams({
                reservesCount: ps._reservesCount,
                liquidationAmount: liquidationAmount,
                auctionRecoveryHealthFactor: ps._auctionRecoveryHealthFactor,
                weth: ADDRESSES_PROVIDER.getWETH(),
                collateralAsset: collateralAsset,
                liquidationAsset: liquidationAsset,
                borrower: borrower,
                liquidator: msg.sender,
                receiveXToken: receivePToken,
                priceOracle: ADDRESSES_PROVIDER.getPriceOracle(),
                priceOracleSentinel: ADDRESSES_PROVIDER.getPriceOracleSentinel(),
                collateralTokenId: 0
            })
        );

        LiquidationLogic.executeLiquidateERC721(
            ps._reserves,
            ps._reservesList,
            ps._usersConfig,
            DataTypes.ExecuteLiquidateParams({
                reservesCount: ps._reservesCount,
                liquidationAmount: maxLiquidationAmount,
                auctionRecoveryHealthFactor: ps._auctionRecoveryHealthFactor,
                weth: ADDRESSES_PROVIDER.getWETH(),
                collateralAsset: collateralAsset,
                liquidationAsset: ADDRESSES_PROVIDER.getWETH(),
                collateralTokenId: collateralTokenId,
                borrower: borrower,
                liquidator: msg.sender,
                receiveXToken: receiveNToken,
                priceOracle: ADDRESSES_PROVIDER.getPriceOracle(),
                priceOracleSentinel: ADDRESSES_PROVIDER.getPriceOracleSentinel()
            })
        );
```

Description

BradMoonUESTC : Not using the return values from these functions can potentially lead to issues with error handling or tracking the state of the tokens being burned or minted. For example, if the burn function call were to fail for some reason, such as insufficient balance or an invalid input, the function would throw an exception, but since the return value is not being used, the exception would not be caught or handled in any way.

It is generally considered good coding practice to handle the return values of functions appropriately, as it helps to ensure the correct functioning of the code and improve its robustness.

Recommendation

BradMoonUESTC : Ensure the return value of external function calls is used. Remove or comment out the unused return function parameters.

Client Response

Acknowledged. We think it's ok.

PSC-6:Use of unified fund withdrawal function

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/misc/AutoCompoundApe.sol#L200**	**Declined**	**Hupixiong3**

Code

```
200:        apeStaking.withdrawApeCoin(amount, receiver);
```

Description

Hupixiong3 : No uniform fund withdrawal function is used,tmp_fix_withdrawFromApeCoinStaking() function and _withdrawFromApeCoinStaking() function money back call interface not consistent.

Recommendation

Hupixiong3 : Use of unified fund withdrawal function

Consider below fix in the `AutoCompoundApe.tmp_fix_withdrawFromApeCoinStaking()` function

```
    function tmp_fix_withdrawFromApeCoinStaking(address receiver)
        external
        onlyOwner
    {
        uint256 amount = 233221446458878461367846 7;
        apeStaking.withdrawSelfApeCoin(amount, receiver);
        (stakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

Client Response

The recommendation is not right. withdrawSelfApecoin don't have a receiver parameter.

PSC-7: `_getTotalPooledApeBalance` may be manipulated by `rewardAmount`

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/misc/AutoCompoundApe.sol#L92-L104	Acknowledged	Secure3

Code

```
92:    function _getTotalPooledApeBalance()
93:        internal
94:        view
95:        override
96:        returns (uint256)
97:    {
98:        uint256 rewardAmount = apeStaking.pendingRewards(
99:            APE_COIN_POOL_ID,
100:            address(this),
101:            0
102:        );
103:        return stakingBalance + rewardAmount + bufferBalance;
104:    }
```

Description

Secure3 : In the `_getTotalPooledApeBalance()` function, the `rewardAmount` is calculated by `apeStaking.pendingRewards()` call.

The `APE_COIN_POOL_ID` is a constant 0, and the parameter `_address` is the address of `AutoCompoundApe` contract

```
  uint256 rewardAmount = apeStaking.pendingRewards(
            APE_COIN_POOL_ID,
            address(this),
            0
        );
 }
```

Zoom into `apeStaking.pendingRewards()` function (in the dependency and a Yuga contract)

```
function pendingRewards(uint256 _poolId, address _address, uint256 _tokenId) external view returns
(uint256) {
        Pool memory pool = pools[_poolId];
        Position memory position = _poolId == 0 ? addressPosition[_address]: nftPosition[_poolId]
[_tokenId];

        (uint256 rewardsSinceLastCalculated,) = rewardsBy(_poolId, pool.lastRewardedTimestampHour,
getPreviousTimestampHour());
        uint256 accumulatedRewardsPerShare = pool.accumulatedRewardsPerShare;

        if (block.timestamp > pool.lastRewardedTimestampHour + SECONDS_PER_HOUR && pool.stakedAmount
!= 0) {
            accumulatedRewardsPerShare = accumulatedRewardsPerShare + rewardsSinceLastCalculated *
APE_COIN_PRECISION / pool.stakedAmount;
        }
        return ((position.stakedAmount * accumulatedRewardsPerShare).toInt256() -
position.rewardsDebt).toUint256() / APE_COIN_PRECISION;
    }
```

Because `APE_COIN_POOL_ID` is 0, position is `addressPosition[_address]`. This is the same cause of the previous hack. `accumulatedRewardsPerShare` does not increase in one block because the `depositApeCoin()` operation calls `updatePool(APECOIN_POOL_ID)` and ensures `pool.lastRewardedTimestampHour` is updated. The `_recipient` parameter in `depositApeCoin()` is set to the `AutoCompoundApe`'s address, so `position.stakedAmount` may be manipulated. This may result in `_getTotalPooledApeBalance()` gets manipulated.

However, since the value of `accumulatedRewardsPerShare` is unclear, this maybe very difficult for hacker to borrow more than the amount staked into the contract within the same block.

Recommendation

Secure3 : To be safe, need a storage variable `rewardAmount` to record the reward amount in the AutoCompoundApe.

Client Response

Acknowledged. We think it's impossible that rewardAmount can be manipulated.

PSC-8: `stakingBalance` should use `realWithdraw` value, not use the input param `amount`

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/misc/AutoCompoundApe.sol#L106-L113	Declined	xfu

Code

```
106:    function _withdrawFromApeCoinStaking(uint256 amount) internal {
107:        uint256 balanceBefore = apeCoin.balanceOf(address(this));
108:        apeStaking.withdrawSelfApeCoin(amount);
109:        uint256 balanceAfter = apeCoin.balanceOf(address(this));
110:        uint256 realWithdraw = balanceAfter - balanceBefore;
111:        stakingBalance -= amount;
112:        bufferBalance += realWithdraw;
113:    }
```

Description

xfu : For trust minimum assumption, suppose `apeStaking.withdrawSelfApeCoin(amount);` execution result is unknown and non-deterministic, using `realWithdraw` is safer than using `amount`

Consider below POC contract

```
    function _withdrawFromApeCoinStaking(uint256 amount) internal {
        uint256 balanceBefore = apeCoin.balanceOf(address(this));
        apeStaking.withdrawSelfApeCoin(amount);
        uint256 balanceAfter = apeCoin.balanceOf(address(this));
        uint256 realWithdraw = balanceAfter - balanceBefore;
        stakingBalance -= amount;
        bufferBalance += realWithdraw;
    }
```

Recommendation

xfu :

```
    function _withdrawFromApeCoinStaking(uint256 amount) internal {
        uint256 balanceBefore = apeCoin.balanceOf(address(this));
        apeStaking.withdrawSelfApeCoin(amount);
        uint256 balanceAfter = apeCoin.balanceOf(address(this));
        uint256 realWithdraw = balanceAfter - balanceBefore;
        stakingBalance -= realWithdraw;
        bufferBalance += realWithdraw;
    }
```

Client Response

Declined. realWithdraw is greater than specified withdraw amount can only be happened when ApeCoin reward is also be withdrawn. The reward amount change cannot be counted in the stakingBalance.

PSC-9: `tmp_fix_withdrawFromApeCoinStaking` should only be allowed once

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/misc/AutoCompoundApe.sol#L195-L202**	**Acknowledged**	**Secure3**

Code

```
    function tmp_fix_withdrawFromApeCoinStaking(address receiver)
        external
        onlyOwner
    {
        uint256 amount = 233221446458878461367846 7;
        apeStaking.withdrawApeCoin(amount, receiver);
        (stakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

Description

Secure3 : the function `tmp_fix_withdrawFromApeCoinStaking` is supposed to only be called one time to set the initial value of `stakingBalance` by calling `withdrawApeCoin` to get the current snapshot the amount of APE staked in the `apeStaking`. As there is no setter function for `stakingBalance`, mistakenlly calling this twice would permanently set the `stakingBalance` valuie to a wrong value.

Recommendation

Secure3 : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Consider below fix in the function

```
    function tmp_fix_withdrawFromApeCoinStaking(address receiver)
        external
        onlyOwner
    {
+       require(stakingBalance == 0, "stakingBalance already set")
        uint256 amount = 2332214464588784613678467;
        apeStaking.withdrawApeCoin(amount, receiver);
        (stakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

Client Response

Acknowledged. We'll remove this function once we got our lending pool recovered.

PSC-10:event duplicate in AutoCompoundApe.sol

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/misc/AutoCompoundApe.sol#L47-L84	Acknowledged	xfu

Code

```
    function deposit(address onBehalf, uint256 amount) external override {
        require(amount > 0, "zero amount");
        uint256 amountShare = getShareByPooledApe(amount);
        if (amountShare == 0) {
            amountShare = amount;
            // permanently lock the first MINIMUM_LIQUIDITY tokens to prevent getPooledApeByShares return 0
            _mint(address(1), MINIMUM_LIQUIDITY);
            amountShare = amountShare - MINIMUM_LIQUIDITY;
        }
        _mint(onBehalf, amountShare);

        _transferTokenIn(msg.sender, amount);
        _harvest();
        _compound();

        emit Transfer(address(0), onBehalf, amount);
        emit Deposit(msg.sender, onBehalf, amount, amountShare);
    }

    /// @inheritdoc IAutoCompoundApe
    function withdraw(uint256 amount) external override {
        require(amount > 0, "zero amount");

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

        _harvest();
        uint256 _bufferBalance = bufferBalance;
        if (amount > _bufferBalance) {
            _withdrawFromApeCoinStaking(amount - _bufferBalance);
        }
        _transferTokenOut(msg.sender, amount);

        _compound();

        emit Transfer(msg.sender, address(0), amount);
        emit Redeem(msg.sender, amount, amountShare);
    }
```

Description

xfu : In deposit and withdraw function, all parameters of transfer have been covered by Deposit and Redeem events, so the transfer event can be removed for saving gas

Consider below POC contract

```
function deposit(address onBehalf, uint256 amount) external override {
        require(amount > 0, "zero amount");
        uint256 amountShare = getShareByPooledApe(amount);
        if (amountShare == 0) {
            amountShare = amount;
            // permanently lock the first MINIMUM_LIQUIDITY tokens to prevent getPooledApeByShares
return 0
            _mint(address(1), MINIMUM_LIQUIDITY);
            amountShare = amountShare - MINIMUM_LIQUIDITY;
        }
        _mint(onBehalf, amountShare);

        _transferTokenIn(msg.sender, amount);
        _harvest();
        _compound();

        emit Transfer(address(0), onBehalf, amount);
        emit Deposit(msg.sender, onBehalf, amount, amountShare);
    }

    /// @inheritdoc IAutoCompoundApe
    function withdraw(uint256 amount) external override {
        require(amount > 0, "zero amount");

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

        _harvest();
        uint256 _bufferBalance = bufferBalance;
        if (amount > _bufferBalance) {
            _withdrawFromApeCoinStaking(amount - _bufferBalance);
        }
        _transferTokenOut(msg.sender, amount);

        _compound();

        emit Transfer(msg.sender, address(0), amount);
        emit Redeem(msg.sender, amount, amountShare);
    }
```

Recommendation

xfu :

```
function deposit(address onBehalf, uint256 amount) external override {
        require(amount > 0, "zero amount");
        uint256 amountShare = getShareByPooledApe(amount);
        if (amountShare == 0) {
            amountShare = amount;
            // permanently lock the first MINIMUM_LIQUIDITY tokens to prevent getPooledApeByShares
return 0
            _mint(address(1), MINIMUM_LIQUIDITY);
            amountShare = amountShare - MINIMUM_LIQUIDITY;
        }
        _mint(onBehalf, amountShare);

        _transferTokenIn(msg.sender, amount);
        _harvest();
        _compound();

        emit Deposit(msg.sender, onBehalf, amount, amountShare);
    }

    /// @inheritdoc IAutoCompoundApe
    function withdraw(uint256 amount) external override {
        require(amount > 0, "zero amount");

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

        _harvest();
        uint256 _bufferBalance = bufferBalance;
        if (amount > _bufferBalance) {
            _withdrawFromApeCoinStaking(amount - _bufferBalance);
        }
        _transferTokenOut(msg.sender, amount);

        _compound();

        emit Redeem(msg.sender, amount, amountShare);
    }
```

Client Response

Acknowledged. We think it's ok.

PSC-11:liquidate the hacker for profit

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/misc/AutoCompoundApe.sol#L101**	**Acknowledged**	**comcat**

Code

```
101:             0
```

Description

comcat : To patch the bug, the following steps were taken:

1. All pToken and debt Token owned by the hackers were transferred to the target address.
2. APE coin was withdrawn from APE_STAKING and the cAPE totalStaked amount was rebalanced. This led to the price of cAPE returning to normal.

As a result, the target address, which now holds all the debt of the hacker, will become insolvent. At this point, a normal user can choose to liquidate it and gain a profit.

Recommendation

comcat : try to deposit asset on behave of the target address, to make it solvent. and avoid liquidation happen.

Client Response

Lending pool is now paused, so liquidation cannot be happen.

PSC-12:tmp_fix_withdrawFromApeCoinStaking may be suffering from front-run attack

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/misc/AutoCompoundApe.sol#L195-L202	Reported	thereksfour

Code

```
    function tmp_fix_withdrawFromApeCoinStaking(address receiver)
        external
        onlyOwner
    {
        uint256 amount = 2332214464588784613678467;
        apeStaking.withdrawApeCoin(amount, receiver);
        (stakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

Description

thereksfour : tmp_fix_withdrawFromApeCoinStaking is aiming to correct exchangeRate by withdrawing excess Apecoin from apeStaking, the issue here is that tmp_fix_withdrawFromApeCoinStaking uses a hard-coded `amount` variable.

```
    function tmp_fix_withdrawFromApeCoinStaking(address receiver)
        external
        onlyOwner
    {
        uint256 amount = 2332214464588784613678467; // @audit: hard-code
        apeStaking.withdrawApeCoin(amount, receiver);
        (stakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

So if an attacker calls apeStaking.depositApeCoin to deposit Apecoin for cAPE before tmp_fix_withdrawFromApeCoinStaking is called, then tmp_fix_withdrawFromApeCoinStaking may not correct the exchangeRate.

More seriously, if the fix transaction is executed in one block and unpause the contract, an attacker may be able to attack again by manipulating exchangeRate.

Consider the following scenario. amount = 23322144645887846136784 67 stakedAmount = 31304781687330337165509 25 _totalShare = 67769049045772886807024 3 Target stakingBalance = stakedAmount - amount = 79826370414424910287245 8 Target exchangeRate = stakingBalance/ _totalShare = 0.85.

The attacker front runs tmp_fix_withdrawFromApeCoinStaking by calling apeStaking.depositApeCoin to deposit Apecoin for cAPE. When tmp_fix_withdrawFromApeCoinStaking is executed, the stakingBalance is larger than expected and the exchangeRate is not corrected.

Recommendation

thereksfour : Consider using the hard-coded exchangeRate instead of the amount

```
    function tmp_fix_withdrawFromApeCoinStaking(address receiver)
        external
        onlyOwner
    {
-       uint256 amount = 2332214464588784613678467;
+      uint256 exchangeRate = 85 * 1e16;
+      uint256 stakedAmount =  apeStaking.addressPosition(address(this));
+      uint256 amount = stakedAmount - _totalShare * 1e18 / exchangeRate ;
        apeStaking.withdrawApeCoin(amount, receiver);
        (stakingBalance, ) = apeStaking.addressPosition(address(this));
    }
```

Client Response

Acknowledged. We'll remove this function once we got our lending pool recovered.

Disclaimer