Competitive Security Assessment

ParaSpace V1.4 P1

Feb 8th, 2023

Secure3

secure3.io

Summary

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

The comprehensive examination and auditing scope includes:

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

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

Overview

Project Detail

Project Name	ParaSpace V1.4 P1
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - dedb08ce2ae56bb5d3e87abba2cd197c17fa498b • final commit - 64037387844cb7b230f9adf9484eada34d39683d
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

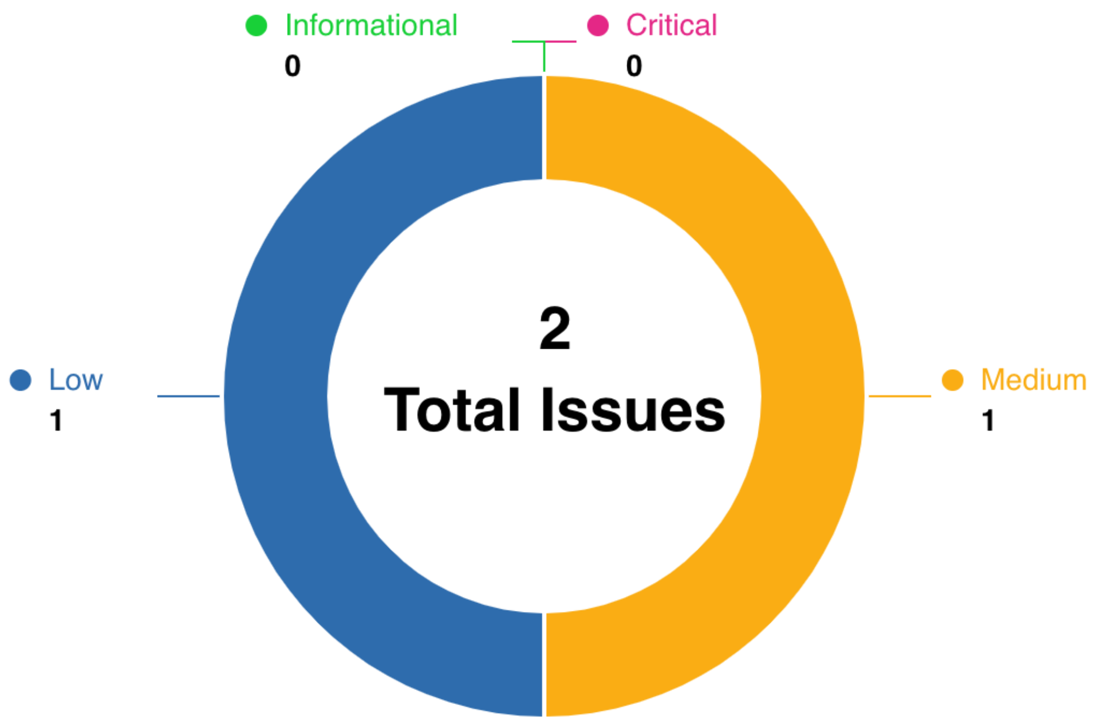
Code Vulnerability Review Summary

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

Audit Scope

File	Commit Hash
contracts/misc/P2PPairStaking.sol	dedb08ce2ae56bb5d3e87abba2cd197c17fa498b

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PSV-1	When BAYC/MAYC/BAKC is liquidated, the liquidated person loses the unclaimed pcApe rewards in P2PPairStaking	Logical	Medium	Mitigated	thereksfour
PSV-2	`P2PPairStaking` contract cannot handle airdrops	Logical	Low	Declined	thereksfour

PSV-1:When BAYC/MAYC/BAKC is liquidated, the liquidated person loses the unclaimed pcApe rewards in P2PPairStaking

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	• **code/contracts/protocol/tokenization/NTokenApeStaking.sol#L85-L91** • **code/contracts/misc/P2PPairStaking.sol#L568-L581**	**Mitigated**	**thereksfour**

Code

```
85:    function _transfer(
86:        address from,
87:        address to,
88:        uint256 tokenId,
89:        bool validate
90:    ) internal override {
91:        ApeStakingLogic.executeUnstakePositionAndRepay(

568:        _depositPCApeShareForUser(
569:            IERC721(_getApeNTokenAddress(order.apeToken)).ownerOf(
570:                order.apeTokenId
571:            ),
572:            rewardShare.percentMul(order.apeShare)
573:        );
574:        _depositPCApeShareForUser(
575:            IERC721(nBakc).ownerOf(order.bakcTokenId),
576:            rewardShare.percentMul(order.bakcShare)
577:        );
578:        _depositPCApeShareForUser(
579:            order.apeCoinOfferer,
580:            rewardShare.percentMul(order.apeCoinShare)
581:        );
```

Description

thereksfour : In P2PPairStaking, rewards are credited to the holder of the NToken only when _claimForMatchedOrderAndCompound is called, and are not collateralized.

```
        _depositPCApeShareForUser(
            IERC721(_getApeNTokenAddress(order.apeToken)).ownerOf(
                order.apeTokenId
            ),
            rewardShare.percentMul(order.apeShare)
        );
        _depositPCApeShareForUser(
            IERC721(nBakc).ownerOf(order.bakcTokenId),
            rewardShare.percentMul(order.bakcShare)
        );
        _depositPCApeShareForUser(
            order.apeCoinOfferer,
            rewardShare.percentMul(order.apeCoinShare)
        );
```

However, when BAYC/MAYC/BAKC is liquidated, _claimForMatchedOrderAndCompound is not called, which means that unclaimed rewards in P2PPairStaking will flow to the liquidator. Unlike this, when ApeStaking in NToken, when BAYC/MAYC/BAKC is liquidated, unstaking is done to avoid the reward from flowing to the liquidator.

```
    function _transfer(
        address from,
        address to,
        uint256 tokenId,
        bool validate
    ) internal override {
        ApeStakingLogic.executeUnstakePositionAndRepay(
```

Recommendation

thereksfour : Consider recording the matching order hash in the NToken of the BAYC/MAYC/BAKC and calling claimForMatchedOrderAndCompound/breakUpMatchedOrder on the order hash when the NToken is liquidated/transferred/burned.

Client Response

This is a known behavior that we are using our off-chain liquidation bot to mitigate. The liquidation bot would either use the unclaimed reward as part of the liquidation, or return any leftover rewards back to the user as part of the standard liquidation process.

PSV-2: `P2PPairStaking` contract cannot handle airdrops

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/libraries/logic/FlashClaimogic.sol#L60-L65 • code/contracts/protocol/tokenization/NToken.sol#L206-L218 • code/contracts/misc/P2PPairStaking.sol#L594-L604	Declined	thereksfour

Code

```
            for (i = 0; i < params.nftTokenIds[index].length; i++) {
                INToken(nTokenAddresses[index]).transferUnderlyingTo(
                    params.receiverAddress,
                    params.nftTokenIds[index][i]
                );
            }

    function transferUnderlyingTo(address target, uint256 tokenId)
        external
        virtual
        override
        onlyPool
        nonReentrant
    {
        IERC721(_underlyingAsset).safeTransferFrom(
            address(this),
            target,
            tokenId
        );
    }

    function _handleApeTransfer(ListingOrder calldata order) internal {
        address currentOwner = IERC721(order.token).ownerOf(order.tokenId);
        if (currentOwner != address(this)) {
            address nTokenAddress = _getApeNTokenAddress(order.token);
            IERC721(order.token).safeTransferFrom(
                nTokenAddress,
                address(this),
                order.tokenId
            );
        }
    }
```

Description

thereksfour : When an order is matched in P2PPairStaking, the user sends the BAYC/MAYC/BAKC to the P2PPairStaking contract, at which point the user still holds the NToken corresponding to the BAYC/MAYC/BAKC. Later, if the user wants to call Pool.flashClaim, in FlashClaimLogic.executeFlashClaim, it will call NToken.transferUnderlyingTo

```
            for (i = 0; i < params.nftTokenIds[index].length; i++) {
                INToken(nTokenAddresses[index]).transferUnderlyingTo(
                    params.receiverAddress,
                    params.nftTokenIds[index][i]
                );
            }
```

In transferUnderlyingTo, the call will fail because BAYC/MAYC/BAKC is not in the NToken contract

```
    function transferUnderlyingTo(address target, uint256 tokenId)
        external
        virtual
        override
        onlyPool
        nonReentrant
    {
        IERC721(_underlyingAsset).safeTransferFrom(
            address(this),
            target,
            tokenId
        );
    }
```

This results in the user not being able to call Pool.flashClaim to claim the airdrop. Of course, the user can break up the order to transfer BAYC/MAYC/BAKC to NToken, but this may cause the cooperative order to be captured by other users.

Recommendation

thereksfour : Consider implementing flashClaim function in P2PPairStaking to allow users to flash claim airdrops without breaking up orders. Or add `rescueERC721`, `rescueERC1155` functions to allow the owner to withdraw the ERC721/ERC1155 airdrops from P2PPairStaking.

Client Response

This is a desire behavior that we are choosing not to support for the time being. Users are free to unmatch to claim airdrop.

Disclaimer