Competitive Security Assessment

ParaSpace-2.0

Aug 28th, 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-2.0
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - 466eeb36dc245d3cdc21f766776cea403b1c19c6 • final commit - 840da50902fd9d62448fbb5b3e8d3c81f8e5d6ce
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

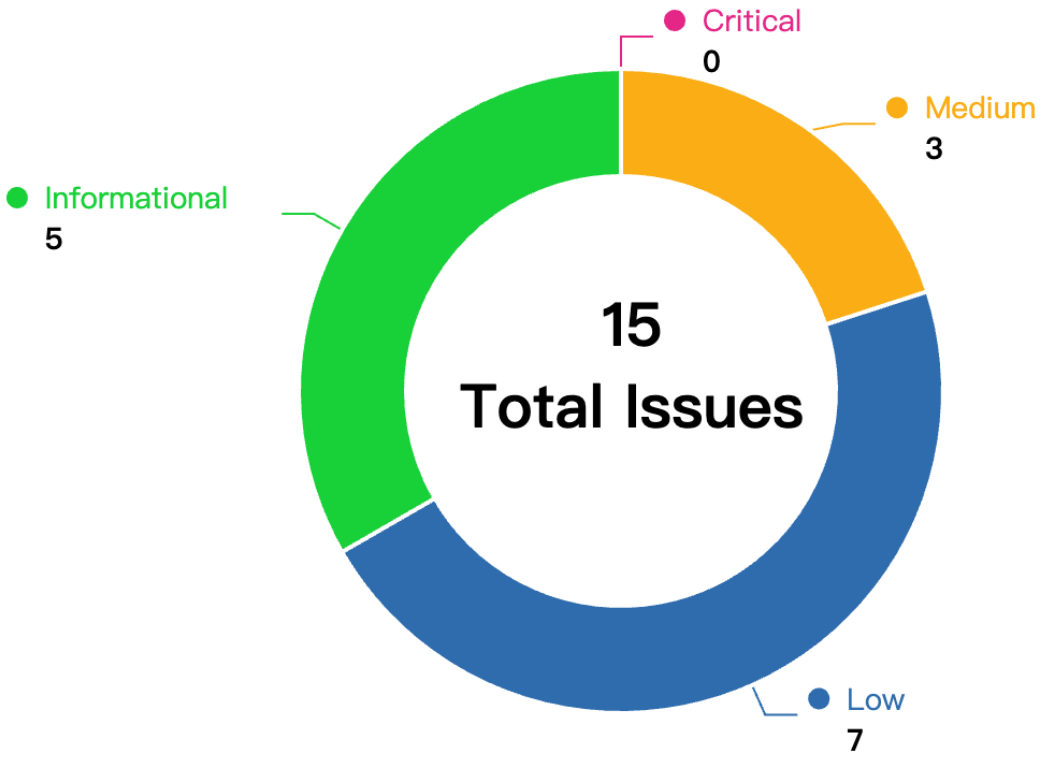
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	3	0	1	0	0	2
Low	7	0	1	1	0	5
Informational	5	0	1	0	0	4

Audit Scope

File	SHA256 Hash
contracts/protocol/libraries/logic/PositionMoverLogic.sol	ef5acdb859e8c161a56a56553bd0038c51834d05f60abcb0675707255fd6c9d3
contracts/protocol/tokenization/PToken.sol	d62effded15d51b531c09ce4d2714a8132e075aceb993590b11849c9f3e0fc9c
contracts/protocol/libraries/logic/ReserveLogic.sol	e800bec8d163c06358365dd7bb2bc037ecfe724c70887242e5422d8934a332fd
contracts/protocol/tokenization/NToken.sol	c51014f1bdd0a453c96b5c82e41d124e41c90b91d46e0fb8c5804eae3c52bbe9
contracts/protocol/pool/PoolPositionMover.sol	d636219136795f61432384bc385258e96f99bcc19c73eb746940e7e020ad249b

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
PAS-1	Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Logical	Medium	Acknowledged	rajatbeladiya
PAS-2	Missing check the return value `lendPool.repay`	Logical	Medium	Declined	comcat, Hupixiong3
PAS-3	Missing `onERC1155BatchReceived` or `onERC721Received()` method	Logical	Medium	Declined	comcat, newway55, n16h7m4r3

PAS-4	claimUnderlying in NToken fail to consider the Punk and Moonbird situation	Logical	Low	Fixed	comcat
PAS-5	Language specific risk in `ReserveLogic::_updateIndexes()` function	Language Specific	Low	Acknowledged	newway55
PAS-6	Bump OZ packages to latest 4.9.3	Logical	Low	Declined	rajatbeladiya
PAS-7	For multiple entry point tokens, PToken.rescueTokens may withdraw the _underlyingAsset.	Logical	Low	Declined	thereksfour
PAS-8	claimUnderlying in the PoolPositionMover maybe underflow for the stETH as underlying	Logical	Low	Declined	comcat
PAS-9	The `ReserveLogic.init()` function parameter `xTokenAddress` need more limit	Logical	Low	Declined	0xac
PAS-10	Oracle Manipulation risk in `PoolPositionMover::movePositionFromParaSpace()` function	Oracle Manipulation	Low	Declined	newway55
PAS-11	Unlocked Pragma Version	Language Specific	Informational	Declined	rajatbeladiya, n16h7m4r3
PAS-12	Gas Optimization: State variables should be cached in stack variables rather than re-reading them from storage	Gas Optimization	Informational	Acknowledged	rajatbeladiya
PAS-13	Gas Optimization: Cache array length outside of loop	Gas Optimization	Informational	Declined	rajatbeladiya
PAS-14	Suggest to ensure the addresses not equal to `0x0` in `transferUnderlyingTo()` function	Code Style	Informational	Declined	0xac
PAS-15	The `transferOnLiquidation()` and `mintToTreasury()` functions can not be called while `POOL` are set as `PoolPositionMover`	Logical	Informational	Declined	0xac

PAS-1:Use safeTransfer/safeTransferFrom instead of transfer/transferFrom

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	rajatbeladiya

Code Reference

- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L298
- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L351

```
298:IERC20(vars.xTokenAddressV1).transferFrom(

351:IERC721(vars.xTokenAddressV1).transferFrom(
```

Description

rajatbeladiya : There are instances of not handling return values of erc20 transfers. It is good to add a `require()` statement that checks the return value of token transfers or to use something like OpenZeppelin's `safeTransfer/safeTransferFrom` unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in contract.

```
IERC20(vars.xTokenAddressV1).transferFrom(
```

`PositionMoverLogic.sol#L298`, `PositionMoverLogic.sol#L351` `transfer/transferFrom` functions are used instead of `safeTransfer/safeTransferFrom` on the following contracts.

Recommendation

rajatbeladiya : Use Openzeppeline's `safeTransfer/safeTransferFrom` instead of `transfer/transferFrom` for the token transfer

Client Response

Acknowledged.confirm, we can improve in the future but not a must for now

PAS-2:Missing check the return value `lendPool.repay`

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	comcat, Hupixiong3

Code Reference

- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L142

```
142:lendPool.repay(loanData.nftAsset, loanData.nftTokenId, borrowAmount);
```

Description

comcat : inside the movePositionFromBendDAO in the PoolPosionMover contract, which will repay the WETH for the user, and get the collateral back to mover contract. after that it will re-collateral the nft to paraspace. however, the repay function in the bendao will return 2 values, which is (vars.repayAmount, !vars.isUpdate). and only when !vars.isUpated == true signals that the nft has been transferred back
Hupixiong3 : Executing the lendPool.repay() function returns a Boolean value of whether the execution was successful.Failure to determine the return value may cause the entire transaction to fail.

Recommendation

comcat : add a requirement to check the return value to make sure that nft is transferred back.

```
function _repayBendDAOPositionLoan(
        ILendPoolLoan lendPoolLoan,
        ILendPool lendPool,
        address weth,
        address xTokenAddress,
        uint256 loanId
    )
        internal
        returns (address nftAsset, uint256 tokenId, uint256 borrowAmount)
    {
        ...

        (, bool isReturnNft) = lendPool.repay(loanData.nftAsset, loanData.nftTokenId, borrowAmount);
        require(isReturnNft, "repay failed");

        (nftAsset, tokenId) = (loanData.nftAsset, loanData.nftTokenId);
    }
```

Hupixiong3 : Judge the return value of lendPool.repay() and deal with it accordingly.

Client Response

declined.INVALID, it'll not cause the transaction to fail

PAS-3:Missing `onERC1155BatchReceived` or `onERC721Received()` method

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	comcat, newway55, n16h7m4r3

Code Reference

- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L142
- code/contracts/protocol/tokenization/NToken.sol#L164-L190
- code/contracts/protocol/tokenization/NToken.sol#L249-L257
- code/contracts/protocol/tokenization/NToken.sol#L259-L267

```
142:lendPool.repay(loanData.nftAsset, loanData.nftTokenId, borrowAmount);

164:function transferUnderlyingTo(
165:        address target,
166:        uint256 tokenId,
167:        DataTypes.TimeLockParams calldata timeLockParams
168:    ) external virtual override onlyPool nonReentrant {
169:        address underlyingAsset = _ERC721Data.underlyingAsset;
170:        if (timeLockParams.releaseTime != 0) {
171:            ITimeLock timeLock = POOL.TIME_LOCK();
172:            uint256[] memory tokenIds = new uint256[](1);
173:            tokenIds[0] = tokenId;
174:            timeLock.createAgreement(
175:                DataTypes.AssetType.ERC721,
176:                timeLockParams.actionType,
177:                underlyingAsset,
178:                tokenIds,
179:                target,
180:                timeLockParams.releaseTime
181:            );
182:            target = address(timeLock);
183:        }
184:
185:        IERC721(underlyingAsset).safeTransferFrom(
186:            address(this),
187:            target,
188:            tokenId
189:        );
190:    }

249:function onERC1155Received(
250:        address,
251:        address,
252:        uint256,
253:        uint256,
254:        bytes calldata
255:    ) external pure override returns (bytes4) {
256:        return this.onERC1155Received.selector;
257:    }

259:function onERC1155BatchReceived(
260:        address,
```

```
261:         address,
262:         uint256[] calldata,
263:         uint256[] calldata,
264:         bytes calldata
265:     ) external pure override returns (bytes4) {
266:         return this.onERC1155BatchReceived.selector;
267:     }
```

Description

comcat : function movePositionFromBendDAO in the PoolPositionMover will repay the WETH for a debt position, and get the collateral NFT back from bendDao. and inside the bendDao repay function, it will transferFrom WETH from msg.sender to its vault. and then will judge whether to transfer the collateral NFT to msg.sender, namely the PoolPositionMover contract.

https://github.com/BendDAO/bend-lending-protocol/blob/5f16c3bee5ccbadd7fefd9cd56ffe793f4751f03/contracts/protocol/LendPoolLoan.sol#L206

however, i check theat the PoolPositionMover doesnt have the onERC721Received function. which will leads to the NFT transfer failed.

newway55 : *Please refer to https://docs.openzeppelin.com/contracts/3.x/api/token/erc721#IERC721Receiver-onERC721Received-address-address-uint256-bytes-* (***onERC721Received***)

OZ documentation states that :

Whenever an IERC721 tokenId token is transferred to this contract via IERC721.safeTransferFrom by operator from from, this function is called.

It must return its Solidity selector to confirm the token transfer. If any other value is returned or the interface is not implemented by the recipient, the transfer will be reverted.

The function `transferUnderlyingTo` from **L185 to 188** is using an arbitrary address `target` for which we don't have checked if there is the check specified by OZ (**receiving address can handle ERC721 via the onERC721Received hook**).

As a matter of fact, the transfer will revert if not implemented on the target address.

POC :

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console2} from "forge-std/Test.sol";
import {MockERC721, MockRecipientWithoutHook, MockTimeLock} from "../mocks";

contract TransferUnderlyingToTest is Test {
    MockERC721 public mockERC721;
    MockRecipientWithoutHook public mockRecipientWithoutHook;
    MockTimeLock public mockTimeLock; // Assuming you also need a mock for TimeLock
    address public testContractAddress; // This would be the address of the contract containing tran
sferUnderlyingTo function

    address public joey = vm.addr(0x1);

    function setUp() public {
        mockERC721 = new MockERC721("Mock Token", "MTK");
        mockRecipientWithoutHook = new MockRecipientWithoutHook();
        mockTimeLock = new MockTimeLock();

        // Setup necessary to make the ERC721 belong to the test contract
        // Assuming mockERC721 has a mint function for simplicity
        mockERC721.mint(testContractAddress, 1);
    }

    function testTransferToRecipientWithHook() public {
        vm.prank(joey);  // Impersonating Joey

        uint256 tokenId = 1;
        DataTypes.TimeLockParams memory timeLockParams;
        timeLockParams.releaseTime = 0; // For simplicity, no timelock

        // Transfer using the function. This would need to be adapted to how you call transferUnderl
yingTo
        transferUnderlyingTo(address(mockERC721), tokenId, timeLockParams);

        // Check if the token was transferred.
        address newOwner = mockERC721.ownerOf(tokenId);
        assertEq(newOwner, address(mockERC721), "Token was not transferred to the recipient with the
hook");
    }

    function testTransferToRecipientWithoutHook() public {
```

```
vm.prank(joey);  // Impersonating Joey

        uint256 tokenId = 1;
        DataTypes.TimeLockParams memory timeLockParams;
        timeLockParams.releaseTime = 0; // For simplicity, no timelock

        vm.expectRevert();  // We expect the transfer to revert because the recipient does not imple
ment the necessary hook
        transferUnderlyingTo(address(mockRecipientWithoutHook), tokenId, timeLockParams);
    }

}
```

n16h7m4r3 : The contract `NToken` allows receiving ERC1155 tokens by implementing token callback handlers `onERC721Received()` and `onERC1155BatchReceived()`. The `NToken` contract does not implement any functions to interact with these tokens i.e. to transfer, burn or approve allowances for these tokens. The tokens received by the contract would be inaccessible and locked in the contract.

Recommendation

comcat : add the onERC721Received function in the PoolPositionMover
newway55 : Ensure that `transferUnderlyingTo` transfers to a valid recipient.
Check the implementation here : https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/IERC721Receiver.sol

```
function transferUnderlyingTo(
    address target,
    uint256 tokenId,
    DataTypes.TimeLockParams calldata timeLockParams
) external virtual override onlyPool nonReentrant {
    address underlyingAsset = _ERC721Data.underlyingAsset;

    if (timeLockParams.releaseTime != 0) {
        ITimeLock timeLock = POOL.TIME_LOCK();
        uint256[] memory tokenIds = new uint256[](1);
        tokenIds[0] = tokenId;
        timeLock.createAgreement(
            DataTypes.AssetType.ERC721,
            timeLockParams.actionType,
            underlyingAsset,
            tokenIds,
            target,
            timeLockParams.releaseTime
        );
        target = address(timeLock);
    }

    // Check if the target address implements onERC721Received
    bytes4 expectedSelector = IERC721Receiver(target).onERC721Received.selector;
    require(
        getERC721ReceivedFunctionSignature(target) == expectedSelector,
        "Target cannot handle ERC721 transfers"
    );

    IERC721(underlyingAsset).safeTransferFrom(
        address(this),
        target,
        tokenId
    );
}

// Helper function to safely call onERC721Received on target and return the function's selector
function getERC721ReceivedFunctionSignature(address _target) public view returns (bytes4) {
    (bool success, bytes memory data) = _target.staticcall(
        abi.encodeWithSignature(
            "onERC721Received(address,address,uint256,bytes)",
```

```
address(0),
            address(0),
            0,
            ""
        )
    );
    if (!success || data.length < 4) {
        return 0;
    }
    bytes4 functionSignature;
    assembly {
        functionSignature := mload(add(data, 32))
    }
    return functionSignature;
}
```

n16h7m4r3 : Consider implementing functions compatible with the respective token standards to allow the contract to access the tokens.

Client Response

declined.INVALID, we have it in PoolCore

PAS-4:claimUnderlying in NToken fail to consider the Punk and Moonbird situation

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	comcat

Code Reference

- code/contracts/protocol/tokenization/NToken.sol#L287
- code/contracts/misc/TimeLock.sol#L167

```
167:function claimMoonBirds(uint256[] calldata agreementIds)

287:ITimeLock(timeLockV1).claim(agreementIds);
```

Description

comcat : in the NToken claimUnderlying, it just call the timelockv1.claim method, however, inside the timelock contract, it has the following different methods for claiming NFT:

- claim
- claimMoonBirds
- claimPunk

which means that for the xTokenAddress is xPunk or xMoonbirds, it should call the corresponding method instead of the `claim` method

Recommendation

comcat : consider the claim situation for the Moonbirds and punk.

Client Response

Fixed.cofirm for MOONBIRD, it's fixed in main branch

PAS-5: Language specific risk in `ReserveLogic::_updateIndexes()` function

Category	Severity	Client Response	Contributor
Language Specific	Low	Acknowledged	newway55

Code Reference

- code/contracts/protocol/libraries/logic/ReserveLogic.sol#L268-L309

```
268:function _updateIndexes(
269:        DataTypes.ReserveData storage reserve,
270:        DataTypes.ReserveCache memory reserveCache
271:    ) internal {
272:        reserveCache.nextLiquidityIndex = reserveCache.currLiquidityIndex;
273:        reserveCache.nextVariableBorrowIndex = reserveCache
274:            .currVariableBorrowIndex;
275:
276:        //only cumulating if there is any income being produced
277:        if (reserveCache.currLiquidityRate != 0) {
278:            uint256 cumulatedLiquidityInterest = MathUtils
279:                .calculateLinearInterest(
280:                    reserveCache.currLiquidityRate,
281:                    reserveCache.reserveLastUpdateTimestamp
282:                );
283:            reserveCache.nextLiquidityIndex = cumulatedLiquidityInterest.rayMul(
284:                reserveCache.currLiquidityIndex
285:            );
286:            reserve.liquidityIndex = reserveCache
287:                .nextLiquidityIndex
288:                .toUint128();
289:
290:            //as the liquidity rate might come only from stable rate loans, we need to ensure
291:            //that there is actual variable debt before accumulating
292:            if (reserveCache.currScaledVariableDebt != 0) {
293:                uint256 cumulatedVariableBorrowInterest = MathUtils
294:                    .calculateCompoundedInterest(
295:                        reserveCache.currVariableBorrowRate,
296:                        reserveCache.reserveLastUpdateTimestamp
297:                    );
298:                reserveCache
299:                    .nextVariableBorrowIndex = cumulatedVariableBorrowInterest
300:                    .rayMul(reserveCache.currVariableBorrowIndex);
301:                reserve.variableBorrowIndex = reserveCache
302:                    .nextVariableBorrowIndex
303:                    .toUint128();
304:            }
305:        }
306:
307:        //solium-disable-next-line
308:        reserve.lastUpdateTimestamp = uint40(block.timestamp);
```

```
309:     }
```

Description

newway55 : *This vulnerability is specifically because of deployment on Arbitrum and we are focusing on Arbitrum/Ethereum differences. (Please refer to : https://docs.arbitrum.io/time)*

`_updateIndexes` function in `ReserveLogic`.sol contract uses `block.timestamp` at the end of the function. We are dealing with interest accrual calculations. A minor accuracy on the block.timestamp can lead to over/under accruing interest. Bots can take advantage of this index update error.

Consequences

- Liquidity Providers : If the liquidity index is higher than it should be (e.g., should be 55 but is 56), liquidity providers will earn more interest than they should.
- Borrowers : If the variable borrow index is higher than it should be, borrowers will owe more interest than they should, making borrowing more expensive.

Recommendation

newway55 : - Use L1 block numbers as a reference: As per the provided documentation, accessing block.number within an Arbitrum smart contract will return a value close to the L1 block number. This can be used as an indirect measure of time. You can consider maintaining a mapping of L1 block numbers to their associated timestamps (from L1). This would provide a more accurate measure of time passage.

- Avoid Direct Dependence on block.timestamp for Short-Term Calculations: Since the inaccuracy of block.timestamp in Arbitrum is more pronounced in the short term, you might consider using an alternative method for short-term calculations. This could involve relying on external oracles or using a combination of block numbers and average block times.

```
// Mapping to store L1 block numbers to their timestamps
mapping(uint256 => uint256) public l1BlockTimestamps;

// Function to update L1 block timestamps. This would ideally be called periodically,
// possibly by an oracle or trusted external source.
function updateL1BlockTimestamp(uint256 l1BlockNumber, uint256 timestamp) external {
    // Only allow trusted sources to update
    require(msg.sender == trustedSource, "Not authorized");
    l1BlockTimestamps[l1BlockNumber] = timestamp;
}

function _updateIndexes(DataTypes.ReserveData storage reserve, DataTypes.ReserveCache memory reserve
Cache) internal {
    reserveCache.nextLiquidityIndex = reserveCache.currLiquidityIndex;
    reserveCache.nextVariableBorrowIndex = reserveCache.currVariableBorrowIndex;

    if (reserveCache.currLiquidityRate != 0) {
        uint256 elapsedTime = l1BlockTimestamps[block.number] - l1BlockTimestamps[reserve.lastUpdate
L1Block];

        uint256 cumulatedLiquidityInterest = MathUtils.calculateLinearInterest(reserveCache.currLiqu
idityRate, elapsedTime);
        reserveCache.nextLiquidityIndex = cumulatedLiquidityInterest.rayMul(reserveCache.currLiquidi
tyIndex);
        reserve.liquidityIndex = reserveCache.nextLiquidityIndex.toUint128();

        if (reserveCache.currScaledVariableDebt != 0) {
            uint256 cumulatedVariableBorrowInterest = MathUtils.calculateCompoundedInterest(reserveC
ache.currVariableBorrowRate, elapsedTime);
            reserveCache.nextVariableBorrowIndex = cumulatedVariableBorrowInterest.rayMul(reserveCac
he.currVariableBorrowIndex);
            reserve.variableBorrowIndex = reserveCache.nextVariableBorrowIndex.toUint128();
        }
    }

    reserve.lastUpdateL1Block = block.number; // Save the last L1 block number for which the indexes
were updated
}
```

Client Response

Acknowledged.confirm, it influences just on the accuracy safe we are safe

PAS-6:Bump OZ packages to latest 4.9.3

Category	Severity	Client Response	Contributor
Logical	Low	Declined	rajatbeladiya

Code Reference

- code/package.json#L41-L42

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

Description

rajatbeladiya : Openzeppeline's contracts installed version is 4.2.0. It can be verify in `package.json`

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

Update the versions of `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` to be the latest in package.json, as they may include important bug fixes.

Recommendation

rajatbeladiya : Update `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` versions from 4.2.0 to 4.9.3.

Client Response

declined.INVALID

PAS-7:For multiple entry point tokens, PToken.rescueTokens may withdraw the _underlyingAsset.

Category	Severity	Client Response	Contributor
Logical	Low	Declined	thereksfour

Code Reference

- code/contracts/protocol/tokenization/PToken.sol#L368-L375

```
368:function rescueTokens(
369:        address token,
370:        address to,
371:        uint256 amount
372:    ) external override onlyPoolAdmin {
373:        require(token != _underlyingAsset, Errors.UNDERLYING_CANNOT_BE_RESCUED);
374:        IERC20(token).safeTransfer(to, amount);
375:    }
```

Description

thereksfour : PToken.rescueTokens is used to rescue tokens that were mistakenly sent to the contract and cannot withdraw the _underlyingAsset from the contract.

```
    function rescueTokens(
        address token,
        address to,
        uint256 amount
    ) external override onlyPoolAdmin {
        require(token != _underlyingAsset, Errors.UNDERLYING_CANNOT_BE_RESCUED);
        IERC20(token).safeTransfer(to, amount);
    }
```

However, for multiple entry point tokens, contracts A and B will both correspond to _underlyingAsset, and when PToken sets contract A as _underlyingAsset, rescueTokens can withdraw that token from contract B.

TrueUSD is an example and has a fairly high market volume: https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2

Recommendation

thereksfour : Validate the _underlyingAsset balance has not changed after the token transfer within the call to PToken::rescueTokens.

Client Response

Declined.INVALID, we support only return back normal ERC20 currencies

PAS-8:claimUnderlying in the PoolPositionMover maybe underflow for the stETH as underlying

Category	Severity	Client Response	Contributor
Logical	Low	Declined	comcat

Code Reference

- code/contracts/protocol/tokenization/PToken.sol#L404
- code/contracts/protocol/pool/PoolPositionMover.sol#L135
- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L311

```
135:reserve.unbacked -= IPToken(reserve.xTokenAddress)

311:reserveV2.unbacked += params

404:IERC20(underlyingAsset).balanceOf(address(this)) -
```

Description

comcat : it stores the unbacked amount in the reserveData, and every time when claimUnderlying, it will deduct the corresponding amount from the unbacked. and inside the PToken.claimUnderlying, if the underlying asset is not cAPE, it will return the balanceOfAfter - balanceOfBefore. however, consider the following scenario, if the underlying asset is stETH, which is supported in paraspace. the balance of stETH will increase as time increase. which means that the:

```
diff0 = balanceAfter0 - balanceBefore0
diff1 = balanceAfter1 - balanceAfter1
diff1 > diff0, when time1 > time0
```

and the

```
reserve.unbacked -= IERC20(underlyingAsset).balanceOf(address(this)) - beforeBalance;
```

maybe underflow because the diff1 increase.

Recommendation

comcat : the unbacked should track the share for the stETH instead of the amount of stETH. as also, the PToken.claimUnderlying should treat stETH special, which should return the share diff instead of the balance diff.

Client Response

declined.INVALID, what we get from v1 timelock can only be the number when we withdraw

PAS-9:The `ReserveLogic.init()` function parameter `xTokenAddress` need more limit

Category	Severity	Client Response	Contributor
Logical	Low	Declined	0xac

Code Reference

- code/contracts/protocol/libraries/logic/ReserveLogic.sol#L137-L157

```
137:function init(
138:        DataTypes.ReserveData storage reserve,
139:        address xTokenAddress,
140:        address variableDebtTokenAddress,
141:        address interestRateStrategyAddress,
142:        address auctionStrategyAddress,
143:        address timeLockStrategyAddress
144:    ) internal {
145:        require(
146:            reserve.xTokenAddress == address(0),
147:            Errors.RESERVE_ALREADY_INITIALIZED
148:        );
149:
150:        reserve.liquidityIndex = uint128(WadRayMath.RAY);
151:        reserve.variableBorrowIndex = uint128(WadRayMath.RAY);
152:        reserve.xTokenAddress = xTokenAddress;
153:        reserve.variableDebtTokenAddress = variableDebtTokenAddress;
154:        reserve.interestRateStrategyAddress = interestRateStrategyAddress;
155:        reserve.auctionStrategyAddress = auctionStrategyAddress;
156:        reserve.timeLockStrategyAddress = timeLockStrategyAddress;
157:    }
```

Description

0xac : The `ReserveLogic.init()` function reuqire `reserve.xTokenAddress` is not a 0x0 address while initializing the `reserve`. However, one `reserve` can be set multiple times if its `reserve.xTokenAddress` is initialized as `address(0)`.

```
function init(
    DataTypes.ReserveData storage reserve,
    address xTokenAddress,
    address variableDebtTokenAddress,
    address interestRateStrategyAddress,
    address auctionStrategyAddress,
    address timeLockStrategyAddress
) internal {
    require(
        reserve.xTokenAddress == address(0),
        Errors.RESERVE_ALREADY_INITIALIZED
    );

    reserve.liquidityIndex = uint128(WadRayMath.RAY);
    reserve.variableBorrowIndex = uint128(WadRayMath.RAY);
    reserve.xTokenAddress = xTokenAddress;
    reserve.variableDebtTokenAddress = variableDebtTokenAddress;
    reserve.interestRateStrategyAddress = interestRateStrategyAddress;
    reserve.auctionStrategyAddress = auctionStrategyAddress;
    reserve.timeLockStrategyAddress = timeLockStrategyAddress;
}
```

Recommendation

0xac : Recommend to ensure the value of input parameter `xTokenAddress` not equal to address(0). It promises that one `reserve` can only initial once.

```
function init(
    DataTypes.ReserveData storage reserve,
    address xTokenAddress,
    address variableDebtTokenAddress,
    address interestRateStrategyAddress,
    address auctionStrategyAddress,
    address timeLockStrategyAddress
) internal {
    require(
        reserve.xTokenAddress == address(0),
        Errors.RESERVE_ALREADY_INITIALIZED
    );
    require(xTokenAddress != address(0));

    reserve.liquidityIndex = uint128(WadRayMath.RAY);
    reserve.variableBorrowIndex = uint128(WadRayMath.RAY);
    reserve.xTokenAddress = xTokenAddress;
    reserve.variableDebtTokenAddress = variableDebtTokenAddress;
    reserve.interestRateStrategyAddress = interestRateStrategyAddress;
    reserve.auctionStrategyAddress = auctionStrategyAddress;
    reserve.timeLockStrategyAddress = timeLockStrategyAddress;
}
```

Client Response

declined.INVALID, but it's done by the governance and we have review team to make sure its correctness

PAS-10:Oracle Manipulation risk in `PoolPositionMover::movePositionFromParaSpace()` function

Category	Severity	Client Response	Contributor
Oracle Manipulation	Low	Declined	newway55

Code Reference

- code/contracts/protocol/pool/PoolPositionMover.sol#L91-L115

```
91:function movePositionFromParaSpace(
92:         DataTypes.ParaSpacePositionMoveInfo calldata moveInfo
93:     ) external nonReentrant {
94:         DataTypes.PoolStorage storage ps = poolStorage();
95:
96:         PositionMoverLogic.executeMovePositionFromParaSpaceV1(
97:             ps,
98:             POOL_V1,
99:             PROTOCOL_DATA_PROVIDER_V1,
100:             CAPE_V1,
101:             CAPE_V2,
102:             DataTypes.ParaSpacePositionMoveParams({
103:                 user: msg.sender,
104:                 cTokens: moveInfo.cTokens,
105:                 cTypes: moveInfo.cTypes,
106:                 cAmountsOrTokenIds: moveInfo.cAmountsOrTokenIds,
107:                 dTokens: moveInfo.dTokens,
108:                 dAmounts: moveInfo.dAmounts,
109:                 to: moveInfo.to,
110:                 reservesCount: ps._reservesCount,
111:                 priceOracle: ADDRESSES_PROVIDER.getPriceOracle(),
112:                 priceOracleSentinel: ADDRESSES_PROVIDER.getPriceOracleSentinel()
113:             })
114:         );
115:     }
```

Description

newway55 : The function `movePositionFromParaSpace` relies on an external oracle for its price data. If this oracle derives its price data from a liquidity pool or a decentralized exchange like Uniswap (which I suppose is the case after

checking all contract dependencies), it can be susceptible to manipulation. An attacker can skew the liquidity pool's asset ratio to manipulate the price temporarily, exploit the lending protocol, and then restore the liquidity pool to its original state for profit.

The issue is present here : `priceOracle: ADDRESSES_PROVIDER.getPriceOracle()`

Recommendation

newway55 : - Utilize multiple oracles and derive a median price. We will add a a function `getMedianPrice functio n` that sources prices from multiple oracles and returns a median, mitigating single oracle manipulation risks. It is less focusing on a single point of failure.

```
function movePositionFromParaSpace(DataTypes.ParaSpacePositionMoveInfo calldata moveInfo) external n
onReentrant {
    DataTypes.PoolStorage storage ps = poolStorage();

    // Use a more reliable price source
    address reliablePriceOracle = getMedianPrice(ADDRESSES_PROVIDER.getAllOracles());

    PositionMoverLogic.executeMovePositionFromParaSpaceV1(
        ps,
        POOL_V1,
        PROTOCOL_DATA_PROVIDER_V1,
        CAPE_V1,
        CAPE_V2,
        DataTypes.ParaSpacePositionMoveParams({
            user: msg.sender,
            cTokens: moveInfo.cTokens,
            cTypes: moveInfo.cTypes,
            cAmountsOrTokenIds: moveInfo.cAmountsOrTokenIds,
            dTokens: moveInfo.dTokens,
            dAmounts: moveInfo.dAmounts,
            to: moveInfo.to,
            reservesCount: ps._reservesCount,
            priceOracle: reliablePriceOracle,
            priceOracleSentinel: ADDRESSES_PROVIDER.getPriceOracleSentinel()
        })
    );
}
```

Client Response

declined.INVALID, oracle is controlled by ParaSpace governance

PAS-11:Unlocked Pragma Version

Category	Severity	Client Response	Contributor
Language Specific	Informational	Declined	rajatbeladiya, n16h7m4r3

Code Reference

- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L2
- code/contracts/protocol/libraries/logic/ReserveLogic.sol#L2
- code/contracts/protocol/tokenization/NToken.sol#L2
- code/contracts/protocol/tokenization/PToken.sol#L2
- code/contracts/protocol/pool/PoolPositionMover.sol#L2

```
2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;
```

Description

rajatbeladiya : `ParaSpace` has files with pragma solidity version number with ^0.8.0. The caret (^) points to unlocked pragma, meaning compiler will use the specified version or above. It's good practice to use a specific solidity version to know compiler bug fixes and optimizations were enabled at the time of compiling the contract.
n16h7m4r3 : Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the floating pragma, i.e. by not using ^ in pragma solidity ^0.8.0, ensures that contracts do not accidentally get deployed using an compiler version with unfixed bugs.

Recommendation

rajatbeladiya : Use specific solidity version (latest version recommended)
n16h7m4r3 : Should lock pragmas to a specific compiler version. Besides, consider the known compiler bugs in the following references and check whether the contracts include those bugs.
Solidity compiler bugs: Solidity repo - known bugs Solidity repo - bugs by version

Client Response

declined.INVALID, it's locked via hardhat.config.ts instead

PAS-12:Gas Optimization: State variables should be cached in stack variables rather than re-reading them from storage

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	rajatbeladiya

Code Reference

- code/contracts/protocol/tokenization/PToken.sol#L135
- code/contracts/protocol/tokenization/PToken.sol#L236

```
135:IERC20(_underlyingAsset).safeTransfer(receiverOfUnderlying, amount);

236:IERC20(_underlyingAsset).safeTransfer(target, amount);
```

Description

rajatbeladiya : The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (100 gas) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses. It saves 100 gas per instance. There are 2 instances of it.

Recommendation

rajatbeladiya : cache state variables in stack variables.

```
address underlyingAsset = _underlyingAsset;
IERC20(underlyingAsset).safeTransfer(receiverOfUnderlying, amount);
```

Client Response

Acknowledged.confirm, we will improve in the future

PAS-13:Gas Optimization: Cache array length outside of loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Declined	rajatbeladiya

Code Reference

- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L87
- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L214
- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L270
- code/contracts/protocol/libraries/logic/PositionMoverLogic.sol#L293
- code/contracts/protocol/pool/PoolPositionMover.sol#L127
- code/contracts/protocol/tokenization/NToken.sol#L134

```
87:for (uint256 index = 0; index < loandIds.length; index++) {

127:for (uint256 index = 0; index < assets.length; index++) {

134:for (uint256 index = 0; index < tokenIds.length; index++) {

214:for (uint256 index = 0; index < params.dTokens.length; index++) {

270:for (uint256 i = 0; i < params.cTokens.length; i++) {

293:params.cAmountsOrTokenIds[i][0] == 0 ||
```

Description

rajatbeladiya : If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Recommendation

rajatbeladiya : Cache array length outside of loop

```
uint256 loanIdsLength = loandIds.length;
for (uint256 index = 0; index < loanIdsLength; index++) {
}
```

Client Response

Declined.INVALID, altough we can cache, but still everytime we need to read it from memory

PAS-14:Suggest to ensure the addresses not equal to `0x0` in `transferUnderlyingTo()` function

Category	Severity	Client Response	Contributor
Code Style	Informational	Declined	0xac

Code Reference

- code/contracts/protocol/tokenization/NToken.sol#L164-L190
- code/contracts/protocol/tokenization/PToken.sol#L216-L237

```
164:function transferUnderlyingTo(
165:        address target,
166:        uint256 tokenId,
167:        DataTypes.TimeLockParams calldata timeLockParams
168:    ) external virtual override onlyPool nonReentrant {
169:        address underlyingAsset = _ERC721Data.underlyingAsset;
170:        if (timeLockParams.releaseTime != 0) {
171:            ITimeLock timeLock = POOL.TIME_LOCK();
172:            uint256[] memory tokenIds = new uint256[](1);
173:            tokenIds[0] = tokenId;
174:            timeLock.createAgreement(
175:                DataTypes.AssetType.ERC721,
176:                timeLockParams.actionType,
177:                underlyingAsset,
178:                tokenIds,
179:                target,
180:                timeLockParams.releaseTime
181:            );
182:            target = address(timeLock);
183:        }
184:
185:        IERC721(underlyingAsset).safeTransferFrom(
186:            address(this),
187:            target,
188:            tokenId
189:        );
190:    }

216:function transferUnderlyingTo(
217:        address target,
218:        uint256 amount,
219:        DataTypes.TimeLockParams calldata timeLockParams
220:    ) external virtual override onlyPool {
221:        if (timeLockParams.releaseTime != 0) {
222:            ITimeLock timeLock = POOL.TIME_LOCK();
223:            uint256[] memory amounts = new uint256[](1);
224:            amounts[0] = amount;
225:
226:            timeLock.createAgreement(
227:                DataTypes.AssetType.ERC20,
228:                timeLockParams.actionType,
229:                _underlyingAsset,
```

```
230:                 amounts,
231:                 target,
232:                 timeLockParams.releaseTime
233:             );
234:             target = address(timeLock);
235:         }
236:         IERC20(_underlyingAsset).safeTransfer(target, amount);
237:     }
```

Description

0xac : The `NToken.transferUnderlyingTo()` function has not ensure the `target` and `timeLock` are not 0 addresses before transfering and calling.

```
function transferUnderlyingTo(
    address target,
    uint256 tokenId,
    DataTypes.TimeLockParams calldata timeLockParams
) external virtual override onlyPool nonReentrant {
    address underlyingAsset = _ERC721Data.underlyingAsset;

    if (timeLockParams.releaseTime != 0) {
        ITimeLock timeLock = POOL.TIME_LOCK();

        // lack of checking

        ...
    }

    // lack of checking
    IERC721(underlyingAsset).safeTransferFrom(
            address(this),
            target,
            tokenId
    );
}
```

The `PToken.transferUnderlyingTo()` function has not ensure the `target` and `timeLock` are not 0 addresses before transfering and calling too.

```
function transferUnderlyingTo(
    address target,
    uint256 amount,
    DataTypes.TimeLockParams calldata timeLockParams
) external virtual override onlyPool {
    if (timeLockParams.releaseTime != 0) {
        ITimeLock timeLock = POOL.TIME_LOCK();
        uint256[] memory amounts = new uint256[](1);
        amounts[0] = amount;

        // lack of checking
        timeLock.createAgreement(
            DataTypes.AssetType.ERC20,
            timeLockParams.actionType,
            _underlyingAsset,
            amounts,
            target,
            timeLockParams.releaseTime
        );
        target = address(timeLock);
    }

    // lack of checking
    IERC20(_underlyingAsset).safeTransfer(target, amount);
}
```

Recommendation

0xac : Recommend to require the address is not a empty address before use.

```
function transferUnderlyingTo(
    address target,
    uint256 tokenId,
    DataTypes.TimeLockParams calldata timeLockParams
) external virtual override onlyPool nonReentrant {
    address underlyingAsset = _ERC721Data.underlyingAsset;

    // add require checking
    require(target != address(0), "Invalid target address");

    if (timeLockParams.releaseTime != 0) {
        ITimeLock timeLock = POOL.TIME_LOCK();

        // add require checking
        require(address(timeLock) != address(0), "Time lock contract not found");

        ...
    }

    ...
}
```

```
function transferUnderlyingTo(
    address target,
    uint256 amount,
    DataTypes.TimeLockParams calldata timeLockParams
) external virtual override onlyPool {
    // add require checking
    require(target != address(0), "Invalid target address");

    if (timeLockParams.releaseTime != 0) {
        ITimeLock timeLock = POOL.TIME_LOCK();
        uint256[] memory amounts = new uint256[](1);
        amounts[0] = amount;

        // add require checking
        require(address(timeLock) != address(0), "Time lock contract not found");
        timeLock.createAgreement(
            DataTypes.AssetType.ERC20,
            timeLockParams.actionType,
            _underlyingAsset,
            amounts,
            target,
            timeLockParams.releaseTime
        );
        target = address(timeLock);
    }

    IERC20(_underlyingAsset).safeTransfer(target, amount);
}
```

Client Response

Declined.INVALID, transferUnderlyingTo is used internally by pool

PAS-15:The `transferOnLiquidation()` and `mintToTreasury()` functions can not be called while `POOL` are set as `PoolPositionMover`

Category	Severity	Client Response	Contributor
Logical	Informational	Declined	0xac

Code Reference

- code/contracts/protocol/tokenization/PToken.sol#L106
- code/contracts/protocol/tokenization/PToken.sol#L117
- code/contracts/protocol/tokenization/PToken.sol#L144
- code/contracts/protocol/tokenization/PToken.sol#L157
- code/contracts/protocol/tokenization/PToken.sol#L220
- code/contracts/protocol/tokenization/PToken.sol#L244
- code/contracts/protocol/tokenization/PToken.sol#L393
- code/contracts/protocol/tokenization/NToken.sol#L88
- code/contracts/protocol/tokenization/NToken.sol#L98
- code/contracts/protocol/tokenization/NToken.sol#L151
- code/contracts/protocol/tokenization/NToken.sol#L168
- code/contracts/protocol/tokenization/NToken.sol#L286

```
88:) external virtual override onlyPool nonReentrant returns (uint64, uint64) {

98:) external virtual override onlyPool nonReentrant returns (uint64, uint64) {

106:) external virtual override onlyPool returns (bool) {

117:) external virtual override onlyPool {

144:onlyPool

151:) external onlyPool nonReentrant {

157:) external virtual override onlyPool {

168:) external virtual override onlyPool nonReentrant {

220:) external virtual override onlyPool {

244:onlyPool

286:) external onlyPool {

393:) external onlyPool returns (uint256) {
```

Description

0xac : These functions are using `onlyPool` modifider in `PToken` and `NToken`.

PToken

- mint: only called by `PoolPositionMover`
- burn: only called by `PoolPositionMover`
- mintToTreasury: only called by `PoolCore`
- transferOnLiquidation: only called by `PoolCore`
- transferUnderlyingTo: only called by `PoolPositionMover`
- handleRepayment: empty, called by `PoolPositionMover` and `PoolCore`
- claimUnderlying: only called by `PoolPositionMover`

NToken

- mint: only called by `PoolPositionMover`
- burn: only called by `PoolPositionMover`
- transferOnLiquidation: only called by `PoolCore`

- transferUnderlyingTo: only called by `PoolPositionMover`
- claimUnderlying: only called by `PoolPositionMover`

While the `POOL` variable of `PToken` and `NToken` are set as `PoolPositionMover`, these functions (`PToken.mintToTreasury()`, `PToken.transferOnLiquidation()` and `NToken.transferOnLiquidation()`) can not be called by `PoolPositionMover`.

Recommendation

0xac : Recommend to add a modifider for `PoolPositionMover`. A part of functions called by `PoolCore` can use the `onlyPool` modifider which require `msg.sender == PoolCore`. Another part of functions called by `PoolPositionMover` can use the `onlyPoolPositionMover` modifider which require `msg.sender == PoolPositionMover`. If the function called by `PoolPositionMover` and `PoolCore`, require `msg.sender == PoolCore || msg.sender == PoolPositionMover`.

Client Response

declined.INVALID, it's diamond proxy

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.