Competitive Security Assessment

ParaSpace TimeLock

Mar 27th, 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 TimeLock
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core/pull/353/ • 7190a44e0244701f588b353ccbd215e045dd015b • 91f948621efe5aa49d6eed2038dfb35b67285de7
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

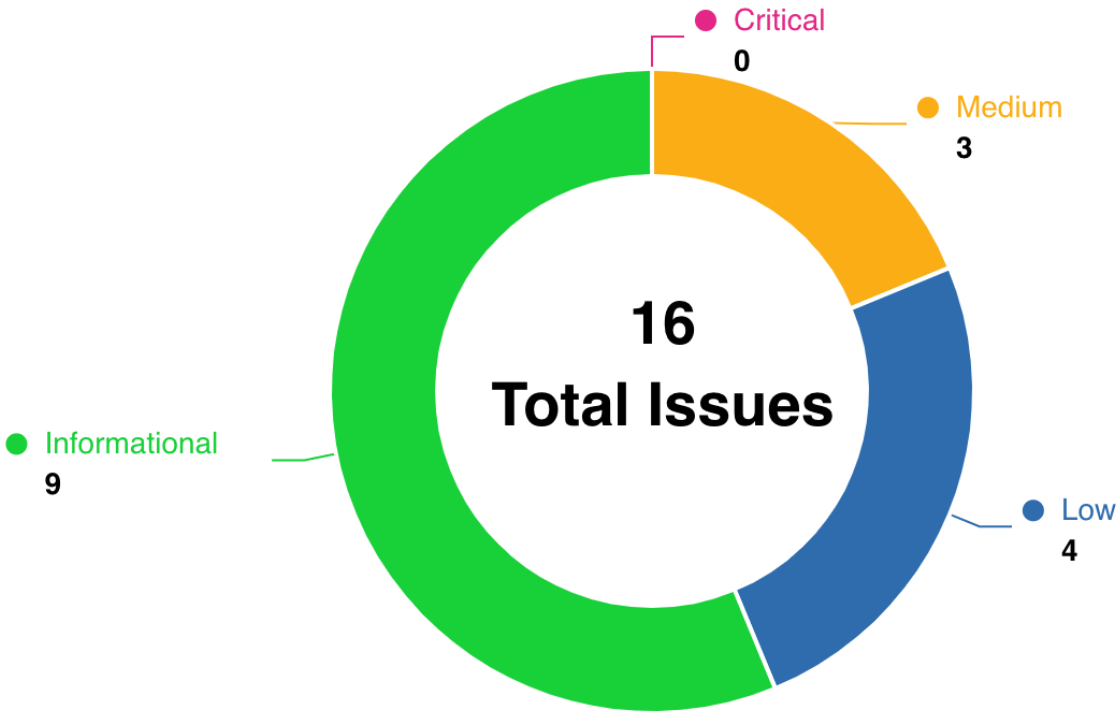
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	3	0	2	1	0	0
Low	4	0	1	3	0	0
Informational	9	0	4	5	0	0

Audit Scope

File	Commit Hash
contracts/interfaces/INToken.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/interfaces/IPToken.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/interfaces/IPoolConfigurator.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/interfaces/IPoolCore.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/interfaces/IPoolParameters.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/interfaces/ITimeLock.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/interfaces/ITimeLockStrategy.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/misc/DefaultTimeLockStrategy.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/misc/TimeLock.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/logic/BorrowLogic.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/logic/FlashClaimLogic.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/logic/GenericLogic.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/logic/LiquidationLogic.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/logic/MarketplaceLogic.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/logic/SupplyLogic.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/libraries/types/DataTypes.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/pool/PoolApeStaking.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/pool/PoolConfigurator.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/pool/PoolCore.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/pool/PoolParameters.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/tokenization/NToken.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/tokenization/NTokenApeStaking.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/tokenization/NTokenBAKC.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/tokenization/NTokenMoonBirds.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/tokenization/PToken.sol	7190a44e0244701f588b353ccbd215e045dd015b
contracts/protocol/tokenization/PTokenSApe.sol	7190a44e0244701f588b353ccbd215e045dd015b

contracts/protocol/tokenization/PYieldToken.sol	**7190a44e0244701f588b353ccbd215e045dd015b**

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PTL-1	Cache array length before for loop	Gas Optimization	Informational	Acknowledged	Hupixiong3
PTL-2	Change `freezeAgreement()` modifier from `public` to `external`	Gas Optimization	Informational	Fixed	Hupixiong3
PTL-3	Lack of sufficient address checks	Code Style	Low	Fixed	BradMoonUESTC
PTL-4	`TimeLock` contract is not compactiable with rebase token, such as stETH	Logical	Medium	Acknowledged	comcat

PTL-5	Miss check in `TimeLock::createAgreement()` can create useless agreements.	Logical	Low	Fixed	Hupixiong3, BradMoonUESTC
PTL-6	Miss emit event for `TimeLock::freezeAgreement` and `TimeLock::freezeAll`	Logical	Informational	Fixed	comcat, jayphbee, Xi_Zi
PTL-7	Missing `onERC721Received` method in `TimeLock` contract	Logical	Medium	Fixed	w2ning, comcat
PTL-8	Missing error message in `onlyXToken` modifier	Code Style	Informational	Fixed	Hupixiong3
PTL-9	Missing event record	Code Style	Informational	Fixed	Kong7ych3
PTL-10	Set a Maximum Threshold for `tokenIdsOrAmounts.length`	Logical	Informational	Acknowledged	Hellobloc
PTL-11	TimeLock - Incomplete function, resulting in the loss of user funds in `TimeLock` contract `claim` function	Logical	Informational	Acknowledged	Xi_Zi
PTL-12	TimeLock.claim/claimMoonBirds cannot specify a recipient	Logical	Informational	Acknowledged	thereksfour
PTL-13	Unused `dummyEventforTypeChain` event	Code Style	Informational	Fixed	Hupixiong3
PTL-14	When users borrow assets, users need to pay the interest incurred while the borrowed assets are locked in the TimeLock	Logical	Medium	Acknowledged	thereksfour
PTL-15	`TimeLock` cannot handle airdrops	Logical	Low	Acknowledged	thereksfour, comcat
PTL-16	immutable parameters without Strict checking in `DefaultTimeLockStrategy`	Code Style	Low	Fixed	Kong7ych3, Xi_Zi, Hellobloc

PTL-1:Cache array length before for loop

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/misc/TimeLock.sol #L138-L144 • code/contracts/misc/TimeLock.sol #L157-L162	Acknowledged	Hupixiong3

Code

```
138:             for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
139:                 erc721.safeTransferFrom(
140:                     address(this),
141:                     agreement.beneficiary,
142:                     agreement.tokenIdsOrAmounts[i]
143:                 );
144:             }

157:         for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
158:             moonBirds.safeTransferWhileNesting(
159:                 address(this),
160:                 agreement.beneficiary,
161:                 agreement.tokenIdsOrAmounts[i]
162:             );
```

Description

Hupixiong3 : In the `claim()` function, the loop condition check can be optimized for gas by caching the array length before the loop

Recommendation

Hupixiong3 : Consider below improvement in the `TimeLock.claim()` function

```
            uint256 len = agreement.tokenIdsOrAmounts.length;
            for (uint256 i = 0; i < len; i++) {
                erc721.safeTransferFrom(
                    address(this),
                    agreement.beneficiary,
                    agreement.tokenIdsOrAmounts[i]
                );

            }
```

Client Response

The gas saving is insignificant given that the call will likely be for 1 agreement or 1 withdrawal

PTL-2:Change `freezeAgreement()` modifier from `public` to `external`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/misc/TimeLock.sol #L166-L171	Fixed	Hupixiong3

Code

```
166:    function freezeAgreement(uint256 agreementId, bool freeze)
167:        public
168:        onlyOwner
169:    {
170:        agreements[agreementId].isFrozen = freeze;
171:    }
```

Description

Hupixiong3 : The function freezeAgreement() is not called by internal functions. external visibility should be preferred to reduce gas consumption when called.

Recommendation

Hupixiong3 : Use external visibility

Client Response

Fixed

PTL-3:Lack of sufficient address checks

Category	Severity	Code Reference	Status	Contributor
Code Style	Low	• code/contracts/protocol/tokenization/NToken.sol#L152-L153 • code/contracts/protocol/pool/PoolParameters.sol#L170 • code/contracts/protocol/pool/PoolConfigurator.sol#L343-L344	Fixed	BradMoonUESTC

Code

```
152:         address target,
153:         uint256 tokenId,

170:         address newStrategyAddress

343:         address asset,
344:         address newRateStrategyAddress
```

Description

BradMoonUESTC : When the address parameter is passed in, there is a lack of sufficient address checks, such as checking whether the address is a zero address For example, wrong strategy address may result in returning 0 value in UiPoolDataProvider

Recommendation

BradMoonUESTC : Add zero address check

Client Response

Fixed

PTL-4: `TimeLock` contract is not compactiable with rebase token, such as stETH

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/tokenization/PTokenStETH.sol#L15 • code/contracts/protocol/tokenization/PToken.sol#L123 • code/contracts/protocol/tokenization/RebasingPToken.sol#L183	Acknowledged	comcat

Code

```
15:contract PTokenStETH is RebasingPToken {

123:                timeLock.createAgreement(

183:    function _burnScaled(
```

Description

comcat : stETH is a rebase token, which means that its balance increases with the passage of time. However, when considering this type of token, the PTokenStETH contract inherits from RebasingPToken, which in turn inherits from PToken. When calling the PToken.burn function, an agreement is created with a specific amount transferred to the timelock contract. The receiver of the agreement will have to wait until the release date to claim their assets. However, when the receiver claims their assets, they can only claim the amount specified in the agreement. This means that any increase in balance during the lock period cannot be claimed by the user. As a result, the user's claimed stETH balance will be smaller than the actual amount, resulting in a loss for the user and a gain for the timelock contract.

Recommendation

comcat : To address this issue, it is suggested that for RebasingPToken, the stETH share in the agreement should be made instead of the actual stETH balance. When the user claims their assets, they can claim back the corresponding amount of their share.

Client Response

We are aware of this and planning to address it in future releases. for now, security is more important so we will keep it the way it is.

PTL-5:Miss check in `TimeLock::createAgreement()` can create useless agreements.

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/misc/TimeLock.sol #L69-L99 • code/contracts/misc/TimeLock.sol #L69-L100	Fixed	Hupixiong3, BradMoonUESTC

Code

```
69:     function createAgreement(
70:         DataTypes.AssetType assetType,
71:         DataTypes.TimeLockActionType actionType,
72:         address asset,
73:         uint256[] calldata tokenIdsOrAmounts,
74:         address beneficiary,
75:         uint48 releaseTime
76:     ) external onlyXToken(asset) returns (uint256) {
77:         uint256 agreementId = agreementCount++;
78:         agreements[agreementId] = Agreement({
79:             assetType: assetType,
80:             actionType: actionType,
81:             asset: asset,
82:             tokenIdsOrAmounts: tokenIdsOrAmounts,
83:             beneficiary: beneficiary,
84:             releaseTime: releaseTime,
85:             isFrozen: false
86:         });
87:
88:         emit AgreementCreated(
89:             agreementId,
90:             assetType,
91:             actionType,
92:             asset,
93:             tokenIdsOrAmounts,
94:             beneficiary,
95:             releaseTime
96:         );
97:
98:         return agreementId;
99:     }
100:
```

```
69:     function createAgreement(
70:         DataTypes.AssetType assetType,
71:         DataTypes.TimeLockActionType actionType,
72:         address asset,
73:         uint256[] calldata tokenIdsOrAmounts,
74:         address beneficiary,
75:         uint48 releaseTime
76:     ) external onlyXToken(asset) returns (uint256) {
77:         uint256 agreementId = agreementCount++;
```

```
78:         agreements[agreementId] = Agreement({
79:             assetType: assetType,
80:             actionType: actionType,
81:             asset: asset,
82:             tokenIdsOrAmounts: tokenIdsOrAmounts,
83:             beneficiary: beneficiary,
84:             releaseTime: releaseTime,
85:             isFrozen: false
86:         });
87:
88:         emit AgreementCreated(
89:             agreementId,
90:             assetType,
91:             actionType,
92:             asset,
93:             tokenIdsOrAmounts,
94:             beneficiary,
95:             releaseTime
96:         );
97:
98:         return agreementId;
99:     }
```

Description

Hupixiong3 : In the createAgreement() function, there is a lack of incoming information validation processing, and the incoming information may be invalid or duplicate.
BradMoonUESTC : The createAgreement function in the TimeLock contract does not have sufficient validation for its input parameters, including release time, token information, and addresses. While the function includes some validation, such as requiring the calling address to be an XToken, it should perform more comprehensive checks on the input parameters to ensure that the contract works as intended and is not vulnerable to attacks.

Recommendation

Hupixiong3 : Adding incoming message validation processing.
BradMoonUESTC : To enhance security, it is recommended to add more input parameter validation to the createAgreement function. Specifically, the contract should validate that the beneficiary address is not a zero address, the release time is in the future, the asset address is not a zero address, the asset type is acceptable, and the tokenIdsOrAmounts array has the appropriate length and contains valid ERC20 token amounts or ERC721 token IDs. It is essential to ensure that these checks are as comprehensive as possible, so that the contract can function correctly and protect users' funds against potential attacks.

Client Response

only our xToken will call creating agreement so it’s okay to keep things simple and no over-do checks. additionally, creating useless agreement is costly and pointless. However, we added minimal checks for extra security.

PTL-6:Miss emit event for `TimeLock::freezeAgreement` and `TimeLock::freezeAll`

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/misc/TimeLock.sol #L166 • code/contracts/misc/TimeLock.sol #L166-L175 • code/contracts/misc/TimeLock.sol #L173-L176	Fixed	comcat, jayphbee, Xi_Zi

Code

```
166:    function freezeAgreement(uint256 agreementId, bool freeze)

166:    function freezeAgreement(uint256 agreementId, bool freeze)
167:        public
168:        onlyOwner
169:    {
170:        agreements[agreementId].isFrozen = freeze;
171:    }
172:
173:    function freezeAllAgreements(bool freeze) external onlyOwner {
174:        frozen = freeze;
175:    }

173:    function freezeAllAgreements(bool freeze) external onlyOwner {
174:        frozen = freeze;
175:    }
176:}
```

Description

comcat : To help off-chain bots monitor the chain status, it is suggested that corresponding events be emitted for government actions, such as freeAgreement and freezeAllAgreements
jayphbee : Users should be notified that agreements are freezed. So event should be emitted when `freezeAgreement` or `freezeAllAgreements` called.

```
    function freezeAgreement(uint256 agreementId, bool freeze)
        public
        onlyOwner
    {
        agreements[agreementId].isFrozen = freeze;
    }

    function freezeAllAgreements(bool freeze) external onlyOwner {
        frozen = freeze;
    }
```

Xi_Zi : The key status of the contract has changed, and it is recommended to add relevant events. The function freezeAllAgreements can suspend key functions of the contract. It is recommended to add an event to this action

Consider below POC contract

```
function freezeAllAgreements(bool freeze) external onlyOwner {
        frozen = freeze;//@audit
    }
```

Recommendation

comcat : add corresponding events.

```
function freezeAgreement(uint256 agreementId, bool freeze)
        public
        onlyOwner
    {
        agreements[agreementId].isFrozen = freeze;
        emit AgreementFreezed(agreementId, freeze);
    }

    function freezeAllAgreements(bool freeze) external onlyOwner {
        frozen = freeze;
        emit AllAgreementsFreezed(freeze);
    }
```

jayphbee : Emit event repectively for `freezeAggreement` and `freezeAllAgreements`.
Xi_Zi : The function freezeAllAgreements can suspend key functions of the contract. It is recommended to add an event to this action

Client Response

Fixed

PTL-7:Missing `onERC721Received` method in `TimeLock` contract

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/misc/TimeLock.sol#L16 • code/contracts/protocol/tokenization/NToken.sol#L122 • code/contracts/misc/TimeLock.sol#L176	Fixed	w2ning, comcat

Code

```
16:contract TimeLock is ITimeLock, OwnableUpgradeable, ReentrancyGuardUpgradeable {

122:                IERC721(_ERC721Data.underlyingAsset).safeTransferFrom(

176:}
```

Description

w2ning : In the `_burn` method of `Ntoken` contract, the NFT could be transferred to `Timelock` contract, but the `Timelock` contract lacks the `onERC721Received` method

```
if (receiverOfUnderlying != address(this)) {
    // address underlyingAsset = _ERC721Data.underlyingAsset;
    if (timeLockParams.releaseTime != 0) {
        ITimeLock timeLock = POOL.TIME_LOCK();
        timeLock.createAgreement(
            DataTypes.AssetType.ERC721,
            timeLockParams.actionType,
            _ERC721Data.underlyingAsset,
            tokenIds,
            receiverOfUnderlying,
            timeLockParams.releaseTime
        );
        receiverOfUnderlying = address(timeLock);
    }

    for (uint256 index = 0; index < tokenIds.length; index++) {
        IERC721(_ERC721Data.underlyingAsset).safeTransferFrom(
            address(this),
            receiverOfUnderlying,
            tokenIds[index]
        );
    }
```

comcat : The timelock contract has been designed to be able to receive all types of tokens, whether they are ERC-20, ERC-721, or ERC-1155. As an example, the NToken, which is the pToken of ERC721/ERC1155, will transfer its underlying token to the timelock contract. However, there is currently no onERC721Received/onERC1155Received interface within the timelock contract. This absence of the required interface could lead to transaction failures and reversions.

Recommendation

w2ning : Add the `onERC721Received` method to the `timelock` contract.
Consider below fix in the `TimeLock` contract

```
import "./IERC721Receiver.sol";

function onERC721Received(
    address,
    address,
    uint256,
    bytes memory
) public override returns (bytes4) {
    return this.onERC721Received.selector;
}
```

comcat : To prevent this, it is recommended to add the corresponding onERC721Received/onERC1155Received interface to the timelock contract.

Client Response

Fixed

PTL-8:Missing error message in `onlyXToken` modifier

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/misc/TimeLock.sol #L56	Fixed	Hupixiong3

Code

```
56:         require(msg.sender == POOL.getReserveXToken(asset));
```

Description

Hupixiong3 : The modifier onlyXToken missing error message.

Recommendation

Hupixiong3 : Add error message.

Client Response

Fixed

PTL-9:Missing event record

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/misc/TimeLock.sol #L170 • code/contracts/misc/TimeLock.sol #L174	Fixed	Kong7ych3

Code

```
170:        agreements[agreementId].isFrozen = freeze;

174:        frozen = freeze;
```

Description

Kong7ych3 : In the TimeLock contract, the owner role can suspend/unsuspend the agreements and timelock respectively through the freezeAgreement and freezeAllAgreements functions, but no event recording is performed.

Recommendation

Kong7ych3 : It is recommended to record events for the modification of sensitive parameters for subsequent community review or self-examination.

Client Response

Fixed

PTL-10:Set a Maximum Threshold for `tokenIdsOrAmounts.length`

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/misc/TimeLock.sol #L64-L94	Acknowledged	Hellobloc

Code

```
64:     function initialize() public initializer {
65:         __Ownable_init();
66:         __ReentrancyGuard_init();
67:     }
68:
69:     function createAgreement(
70:         DataTypes.AssetType assetType,
71:         DataTypes.TimeLockActionType actionType,
72:         address asset,
73:         uint256[] calldata tokenIdsOrAmounts,
74:         address beneficiary,
75:         uint48 releaseTime
76:     ) external onlyXToken(asset) returns (uint256) {
77:         uint256 agreementId = agreementCount++;
78:         agreements[agreementId] = Agreement({
79:             assetType: assetType,
80:             actionType: actionType,
81:             asset: asset,
82:             tokenIdsOrAmounts: tokenIdsOrAmounts,
83:             beneficiary: beneficiary,
84:             releaseTime: releaseTime,
85:             isFrozen: false
86:         });
87:
88:         emit AgreementCreated(
89:             agreementId,
90:             assetType,
91:             actionType,
92:             asset,
93:             tokenIdsOrAmounts,
94:             beneficiary,
```

Description

Hellobloc : The `time-Lock` contract has a `claim()` operation based on `tokenIdsOrAmounts` for the retrieval of the unlocked assets, and in which the `tokenIdsOrAmounts` are traversed, but there is no check on the `length` of `tokenIdsOrAmounts`.

```
function claim(uint256[] calldata agreementIds) external nonReentrant {
        ...
        for (uint256 index = 0; index < agreementIds.length; index++) {
               ...
            } else if (agreement.assetType == DataTypes.AssetType.ERC721) {
                IERC721 erc721 = IERC721(agreement.asset);
                for (
                    uint256 i = 0;
                    i < agreement.tokenIdsOrAmounts.length;
                    i++
                ) {
                    erc721.safeTransferFrom(
                        ...
                    );
                }
            }
        }
    }
```

This may lead to failed transactions not being revert early enough to reduce `Gas loss`.

Further considering the `GasLimit` of eth's transactions, the irregular `TimeLock` call may lead to the `creationAgreement()` transaction consuming much less Gas than `claim()`, which may lead to the permanent freezing of assets. (Because `tokenIdsOrAmounts` in `claim` may cause the transaction to reach `GasLimit`, while `createAgreement` is normal.).

Recommendation

Hellobloc : It is recommended to add a `maximum-threshold-check` for the `tokenIdsOrAmounts.length` in `createAgreement`.

Further, the same check can be added before calling `createAgreement` to revert transactions that reach the upper limit of Gas as early as possible to reduce Gas loss.

Client Response

Good idea in general, However, only our xToken will call creating agreement so it’s okay to keep things simple and no over-do checks. additionally, creating useless agreement is costly and pointless.

PTL-11:TimeLock - Incomplete function, resulting in the loss of user funds in `TimeLock` contract `claim` function

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/misc/TimeLock.sol #L128-L147	Acknowledged	Xi_Zi

Code

```
128:     function claim(uint256 agreementId) external nonReentrant {
129:         Agreement memory agreement = _validateAndDeleteAgreement(agreementId);
130:
131:         if (agreement.assetType == DataTypes.AssetType.ERC20) {
132:             IERC20(agreement.asset).safeTransfer(
133:                 agreement.beneficiary,
134:                 agreement.tokenIdsOrAmounts[0]
135:             );
136:         } else if (agreement.assetType == DataTypes.AssetType.ERC721) {
137:             IERC721 erc721 = IERC721(agreement.asset);
138:             for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
139:                 erc721.safeTransferFrom(
140:                     address(this),
141:                     agreement.beneficiary,
142:                     agreement.tokenIdsOrAmounts[i]
143:                 );
144:             }
145:         }
146:     }
147:
```

Description

Xi_Zi : According to the design document description， *TimeLock Contract allows the admin to submit time-locked agreements for various token types (ERC20, ERC721, and ERC1155)*. However, in the contract claim function, only ERC20 and ERC721 tokens are processed but ERC1155 type tokens are not processed, which may cause the user's ECR1155 type tokens to not be withdrawn, affecting the security of user funds
https://parallelfinance.notion.site/TimeLock-on-ParaSpace-Withdrawals-and-Borrows-dc0831edc6314ea18f8695a7c40d7da4

```
function claim(uint256 agreementId) external nonReentrant {
        Agreement memory agreement = _validateAndDeleteAgreement(agreementId);

        if (agreement.assetType == DataTypes.AssetType.ERC20) {
            IERC20(agreement.asset).safeTransfer(
                agreement.beneficiary,
                agreement.tokenIdsOrAmounts[0]
            );
        } else if (agreement.assetType == DataTypes.AssetType.ERC721) {
            IERC721 erc721 = IERC721(agreement.asset);
            for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
                erc721.safeTransferFrom(
                    address(this),
                    agreement.beneficiary,
                    agreement.tokenIdsOrAmounts[i]
                );
            }
        }//@audit
    }
```

Recommendation

Xi_Zi : According to the design document, add the processing of the token of type ERC155 to the function claim.

Client Response

ERC1155 will be implemented in the future. for now, we don’t have a use case for it.

PTL-12:TimeLock.claim/claimMoonBirds cannot specify a recipient

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/misc/TimeLock.sol #L128-L164	Acknowledged	thereksfour

Code

```
128:     function claim(uint256 agreementId) external nonReentrant {
129:         Agreement memory agreement = _validateAndDeleteAgreement(agreementId);
130:
131:         if (agreement.assetType == DataTypes.AssetType.ERC20) {
132:             IERC20(agreement.asset).safeTransfer(
133:                 agreement.beneficiary,
134:                 agreement.tokenIdsOrAmounts[0]
135:             );
136:         } else if (agreement.assetType == DataTypes.AssetType.ERC721) {
137:             IERC721 erc721 = IERC721(agreement.asset);
138:             for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
139:                 erc721.safeTransferFrom(
140:                     address(this),
141:                     agreement.beneficiary,
142:                     agreement.tokenIdsOrAmounts[i]
143:                 );
144:             }
145:         }
146:     }
147:
148:     function claimMoonBirds(uint256 agreementId) external nonReentrant {
149:         Agreement memory agreement = _validateAndDeleteAgreement(agreementId);
150:
151:         require(
152:             agreement.assetType == DataTypes.AssetType.ERC721,
153:             "wrong asset type"
154:         );
155:
156:         IMoonBird moonBirds = IMoonBird(agreement.asset);
157:         for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
158:             moonBirds.safeTransferWhileNesting(
159:                 address(this),
160:                 agreement.beneficiary,
161:                 agreement.tokenIdsOrAmounts[i]
162:             );
163:         }
164:     }
```

Description

thereksfour : TimeLock.claim/claimMoonBirds is used to claim assets from TimeLock, the issue here is that the contract only allows the assets to be sent to the `beneficiary` address and does not allow the claimant to specify the recipient,

which in some extreme cases may result in the user not being able to claim the assets.

```
    function claim(uint256 agreementId) external nonReentrant {
        Agreement memory agreement = _validateAndDeleteAgreement(agreementId);

        if (agreement.assetType == DataTypes.AssetType.ERC20) {
            IERC20(agreement.asset).safeTransfer(
                agreement.beneficiary,
                agreement.tokenIdsOrAmounts[0]
            );
        } else if (agreement.assetType == DataTypes.AssetType.ERC721) {
            IERC721 erc721 = IERC721(agreement.asset);
            for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
                erc721.safeTransferFrom(
                    address(this),
                    agreement.beneficiary,
                    agreement.tokenIdsOrAmounts[i]
                );
            }
        }
    }

    function claimMoonBirds(uint256 agreementId) external nonReentrant {
        Agreement memory agreement = _validateAndDeleteAgreement(agreementId);

        require(
            agreement.assetType == DataTypes.AssetType.ERC721,
            "wrong asset type"
        );

        IMoonBird moonBirds = IMoonBird(agreement.asset);
        for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
            moonBirds.safeTransferWhileNesting(
                address(this),
                agreement.beneficiary,
                agreement.tokenIdsOrAmounts[i]
            );
        }
    }
```

For ERC20 tokens such as USDC, if the beneficiary address is added to the USDC blacklist during the asset lock, these USDC tokens will not be available for claiming. For ERC721 tokens, if the beneficiary is a smart contract and is upgraded

during the asset lock and does not implement the onerc721received function, then these ERC721 tokens will not be available for claiming.

Recommendation

thereksfour :

```
-   function claim(uint256 agreementId) external nonReentrant {
+   function claim(uint256 agreementId, address to) external nonReentrant {
        Agreement memory agreement = _validateAndDeleteAgreement(agreementId);

        if (agreement.assetType == DataTypes.AssetType.ERC20) {
            IERC20(agreement.asset).safeTransfer(
-               agreement.beneficiary,
+               to,
                agreement.tokenIdsOrAmounts[0]
            );
        } else if (agreement.assetType == DataTypes.AssetType.ERC721) {
            IERC721 erc721 = IERC721(agreement.asset);
            for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
                erc721.safeTransferFrom(
                    address(this),
-                   agreement.beneficiary,
+                   to,
                    agreement.tokenIdsOrAmounts[i]
                );
            }
        }
    }

-   function claimMoonBirds(uint256 agreementId) external nonReentrant {
+   function claimMoonBirds(uint256 agreementId, address to) external nonReentrant {

        Agreement memory agreement = _validateAndDeleteAgreement(agreementId);

        require(
            agreement.assetType == DataTypes.AssetType.ERC721,
            "wrong asset type"
        );

        IMoonBird moonBirds = IMoonBird(agreement.asset);
        for (uint256 i = 0; i < agreement.tokenIdsOrAmounts.length; i++) {
            moonBirds.safeTransferWhileNesting(
                address(this),
-               agreement.beneficiary,
+               to,
                agreement.tokenIdsOrAmounts[i]
            );
        }
    }
```

Client Response

Acknowledged

PTL-13:Unused `dummyEventforTypeChain` event

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/protocol/libraries/logic/GenericLogic.sol#L60	Fixed	Hupixiong3

Code

```
60:    event dummyEventforTypeChain();
```

Description

Hupixiong3 : There are unused dummyEventforTypeChain() event in the GenericLogic contract, redundant code that causes unnecessary gas consumption and makes code maintenance more difficult.

Recommendation

Hupixiong3 : Delete dummyEventforTypeChain() event.

Client Response

Fixed

PTL-14:When users borrow assets, users need to pay the interest incurred while the borrowed assets are locked in the TimeLock

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/libraries/logic/BorrowLogic.sol#L54-L120	Acknowledged	thereksfour

Code

```
54:     function executeBorrow(
55:         mapping(address => DataTypes.ReserveData) storage reservesData,
56:         mapping(uint256 => address) storage reservesList,
57:         DataTypes.UserConfigurationMap storage userConfig,
58:         DataTypes.ExecuteBorrowParams memory params
59:     ) public {
60:         DataTypes.ReserveData storage reserve = reservesData[params.asset];
61:         DataTypes.ReserveCache memory reserveCache = reserve.cache();
62:
63:         reserve.updateState(reserveCache);
64:
65:         ValidationLogic.validateBorrow(
66:             reservesData,
67:             reservesList,
68:             DataTypes.ValidateBorrowParams({
69:                 reserveCache: reserveCache,
70:                 userConfig: userConfig,
71:                 asset: params.asset,
72:                 userAddress: params.onBehalfOf,
73:                 amount: params.amount,
74:                 reservesCount: params.reservesCount,
75:                 oracle: params.oracle,
76:                 priceOracleSentinel: params.priceOracleSentinel
77:             })
78:         );
79:
80:         bool isFirstBorrowing = false;
81:
82:         (
83:             isFirstBorrowing,
84:             reserveCache.nextScaledVariableDebt
85:         ) = IVariableDebtToken(reserveCache.variableDebtTokenAddress).mint(
86:             params.user,
87:             params.onBehalfOf,
88:             params.amount,
89:             reserveCache.nextVariableBorrowIndex
90:         );
91:
92:         if (isFirstBorrowing) {
93:             userConfig.setBorrowing(reserve.id, true);
94:         }
95:
```

```
96:         reserve.updateInterestRates(
97:             reserveCache,
98:             params.asset,
99:             0,
100:             params.releaseUnderlying ? params.amount : 0
101:         );
102:
103:         if (params.releaseUnderlying) {
104:             DataTypes.TimeLockParams memory timeLockParams = GenericLogic
105:                 .calculateTimeLockParams(
106:                     reserve,
107:                     DataTypes.TimeLockFactorParams({
108:                         assetType: DataTypes.AssetType.ERC20,
109:                         asset: params.asset,
110:                         amount: params.amount
111:                     })
112:                 );
113:             timeLockParams.actionType = DataTypes.TimeLockActionType.BORROW;
114:
115:             IPToken(reserveCache.xTokenAddress).transferUnderlyingTo(
116:                 params.user,
117:                 params.amount,
118:                 timeLockParams
119:             );
120:         }
```

Description

thereksfour : When users borrow assets, in the executeBorrow function, the assets are sent to TimeLock for locking, and the user can only claim them after the release time.

```
        if (params.releaseUnderlying) {
            DataTypes.TimeLockParams memory timeLockParams = GenericLogic
                .calculateTimeLockParams(
                    reserve,
                    DataTypes.TimeLockFactorParams({
                        assetType: DataTypes.AssetType.ERC20,
                        asset: params.asset,
                        amount: params.amount
                    })
                );
            timeLockParams.actionType = DataTypes.TimeLockActionType.BORROW;

            IPToken(reserveCache.xTokenAddress).transferUnderlyingTo(
                params.user,
                params.amount,
                timeLockParams
            );
        }
```

The issue here is that the user does not get the assets, but the borrowed assets have already started to accrue interest, and the user will have to pay for the interest incurred during the lock period.

This will have minimal impact on long term borrowing, but will have a high impact on short term and large amount borrowing.

Considering the 24-hour lock-in period, if a user wants to borrow an asset and repay it after 48 hours, the user can hold the asset for a maximum of 24 hours, but the user pays the interest for 48 hours.

Recommendation

thereksfour : Consider rewarding assets locked in TimeLock based on the lock type, lock time, and lock amount, thus offsetting the user's loss due to the asset lock.

Client Response

This is the intended behaviour. Thanks for the suggestion.

PTL-15: `TimeLock` cannot handle airdrops

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/misc/TimeLock.sol #L16-L17 • code/contracts/misc/TimeLock.sol #L16	Acknowledged	thereksfour, comcat

Code

```
16:contract TimeLock is ITimeLock, OwnableUpgradeable, ReentrancyGuardUpgradeable {
```

```
16:contract TimeLock is ITimeLock, OwnableUpgradeable, ReentrancyGuardUpgradeable {
17:     using GPv2SafeERC20 for IERC20;
```

Description

thereksfour : When the user withdraws the underlying NFT from the NToken, the NFT is sent to TimeLock for locking and can only be withdrawn after the release time. The issue here is that airdrops cannot be handled when the NFT is locked in TimeLock.

1. For push airdrops, the airdrops are sent directly to the holder, the TimeLock contract, and since the TimeLock contract does not have functions like rescueERC721/rescueERC1155 to withdraw the airdrops from the contract, these airdrops are locked in the contract.
2. for pull airdrops, the TimeLock contract does not implement a flashclaim-like function to allow users to claim the airdrops.
   comcat : According to the Moonbirds website, holders of Moonbirds NFTs can become eligible for additional benefits by "nesting." These benefits can include exclusive airdrops and perks, which may be available to all holders or only those with specific traits.

However, if a user withdraws their Moonbirds NFT, it will be transferred to the timelock contract. If the NFT is then eligible for an airdrop, the airdropped token will belong to the timelock contract instead of the original owner. The token will also be locked inside the timelock contract forever, and there will be no way to retrieve it.

Recommendation

thereksfour : Consider implementing rescueERC721/rescueERC1155 and flashclaim functions to handle airdrops
comcat : To address this issue, it is recommended that a "rescue" method be added, which would only be accessible by the owner. This method would properly handle any airdropped tokens and ensure that they are not considered legitimate assets stored in the timelock. This would help to limit the power of the "rescue" method and prevent any misuse.

```
function rescue(address token, uint tokenId, uint amount, address receiver, uint type) onlyOwner
public {
         for (uint i = 0; i < assets.length; i++) {
              require(token != assets[i], "can not withdraw legitimate token");
          }
          if (type == DataTypes.AssetType.ERC20) {
              ...
         } else if (type == DataTypes.AssetType.ERC721) {
             ...
         }

}
```

Client Response

We will handle in V2

PTL-16:immutable parameters without Strict checking in `DefaultTimeLockStrategy`

Category	Severity	Code Reference	Status	Contributor
Code Style	Low	- code/contracts/misc/DefaultTimeLockStrategy.sol#L29-L50 • code/contracts/misc/DefaultTimeLockStrategy.sol#L29-L50 • code/contracts/misc/DefaultTimeLockStrategy.sol#L31-L52	Fixed	Kong7ych3, Xi_Zi, Hellobloc

Code

```
29:     constructor(
30:         address pool,
31:         uint256 minThreshold,
32:         uint256 midThreshold,
33:         uint48 minWaitTime,
34:         uint48 midWaitTime,
35:         uint48 maxWaitTime,
36:         uint256 maxPoolPeriodRate,
37:         uint48 maxPoolPeriodWaitTime,
38:         uint256 period
39:     ) {
40:         POOL = pool;
41:         MIN_THRESHOLD = minThreshold;
42:         MID_THRESHOLD = midThreshold;
43:
44:         MIN_WAIT_TIME = minWaitTime;
45:         MID_WAIT_TIME = midWaitTime;
46:         MAX_WAIT_TIME = maxWaitTime;
47:         MAX_POOL_PERIOD_RATE = maxPoolPeriodRate;
48:         POOL_PERIOD_RATE_WAIT_TIME = maxPoolPeriodWaitTime;
49:         PERIOD = period;
50:     }
```

```
29:     constructor(
30:         address pool,
31:         uint256 minThreshold,
32:         uint256 midThreshold,
33:         uint48 minWaitTime,
34:         uint48 midWaitTime,
35:         uint48 maxWaitTime,
36:         uint256 maxPoolPeriodRate,
37:         uint48 maxPoolPeriodWaitTime,
38:         uint256 period
39:     ) {
40:         POOL = pool;
41:         MIN_THRESHOLD = minThreshold;
42:         MID_THRESHOLD = midThreshold;
43:
44:         MIN_WAIT_TIME = minWaitTime;
45:         MID_WAIT_TIME = midWaitTime;
46:         MAX_WAIT_TIME = maxWaitTime;
47:         MAX_POOL_PERIOD_RATE = maxPoolPeriodRate;
```

```
48:         POOL_PERIOD_RATE_WAIT_TIME = maxPoolPeriodWaitTime;
49:         PERIOD = period;
50:     }

31:         uint256 minThreshold,
32:         uint256 midThreshold,
33:         uint48 minWaitTime,
34:         uint48 midWaitTime,
35:         uint48 maxWaitTime,
36:         uint256 maxPoolPeriodRate,
37:         uint48 maxPoolPeriodWaitTime,
38:         uint256 period
39:     ) {
40:         POOL = pool;
41:         MIN_THRESHOLD = minThreshold;
42:         MID_THRESHOLD = midThreshold;
43:
44:         MIN_WAIT_TIME = minWaitTime;
45:         MID_WAIT_TIME = midWaitTime;
46:         MAX_WAIT_TIME = maxWaitTime;
47:         MAX_POOL_PERIOD_RATE = maxPoolPeriodRate;
48:         POOL_PERIOD_RATE_WAIT_TIME = maxPoolPeriodWaitTime;
49:         PERIOD = period;
50:     }
51:
52:     function resetPeriodLimit() internal {
```

Description

Kong7ych3 : In the DefaultTimeLockStrategy contract, the MIN_THRESHOLD, MID_THRESHOLD, MIN_WAIT_TIME, MID_WAIT_TIME, MAX_WAIT_TIME, POOL_PERIOD_RATE_WAIT_TIME, PERIOD and POOL parameters are all immutable. They are only set when the contract is initialized, so it is necessary to check whether the parameters passed in when the contract is initialized meet expectations. If the parameters passed in are wrong, the contract will be abandoned.

Xi_Zi : The contract does not verify the initialization of WaitTime and Threshold in order of size. If minWaitTime, minWaitTime and maxWaitTime are not set according to minWaitTime<minWaitTime<maxWaitTime, minThreshold, midThreshold is not set in the order of the size of minThreshold<midThreshold, and since the above variables cannot be modified by related functions after initialization, if the initialization is not verified, once the setting is wrong, calculateTimeLockParams function result may be calculated incorrectly.

```
constructor(
        address pool,
        uint256 minThreshold,
        uint256 midThreshold,
        uint48 minWaitTime,
        uint48 midWaitTime,
        uint48 maxWaitTime,
        uint256 maxPoolPeriodRate,
        uint48 maxPoolPeriodWaitTime,
        uint256 period
    ) {
        POOL = pool;
        MIN_THRESHOLD = minThreshold;//@audit
        MID_THRESHOLD = midThreshold;//@audit

        MIN_WAIT_TIME = minWaitTime;//@audit
        MID_WAIT_TIME = minWaitTime;//@audit
        MAX_WAIT_TIME = maxWaitTime;//@audit
        MAX_POOL_PERIOD_RATE = maxPoolPeriodRate;
        POOL_PERIOD_RATE_WAIT_TIME = maxPoolPeriodWaitTime;
        PERIOD = period;
    }
```

Hellobloc : The initialization of the `releaseTime` related parameters is done in the `constructor` of the `DefaultTimeLockStrategy`, which is used to set the `releaseTime`. the initialization of these parameters is very important, considering that the `releaseTime` setting will affect the freezing time of `assets`.

However, there is no `basic-check` in the `constructor` to ensure that the parameters are not incorrectly set to unreasonable values.

Recommendation

Kong7ych3 : It is recommended to check whether the immutable parameters passed in for initialization meet expectations.

Consider below fix in the `DefaultTimeLockStrategy.constructor()` function

```
    constructor(
        address pool,
        uint256 minThreshold,
        uint256 midThreshold,
        uint48 minWaitTime,
        uint48 midWaitTime,
        uint48 maxWaitTime,
        uint256 maxPoolPeriodRate,
        uint48 maxPoolPeriodWaitTime,
        uint256 period
    ) {
        // Strict checking of immutable parameters
        require(minThreshold > 0 && midThreshold > minThreshold);
        require(minWaitTime > 0 && midWaitTime > minWaitTime && maxWaitTime > midWaitTime);
        require(pool != address(0) && maxPoolPeriodRate != 0 && maxPoolPeriodWaitTime != 0 && period
!= 0);

        ...
    }
```

Xi_Zi : It is recommended to verify Threshold and WaitTime when the contract is initialized to ensure that they are initialized in the correct size order.

Hellobloc : We recommend adding the following checks for `releaseTime` related parameters.

```
require(minThreshold > MIN_THRESHOLD && midThreshold > minThreshold && MAX_THRESHOLD >
midThreshold);
require(minWaitTime > MIN_WAIT_TIME && midWaitTime > minWaitTime &&  maxWaitTime > midWaitTime &&
MAX_WAIT_TIME > maxWaitTime);
require(MAX_POOL_PERIOD_WAIT_TIME > poolPeriodWaitTime);
require(MAX_PERIOD >= period);
```

Client Response

Fixed

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.