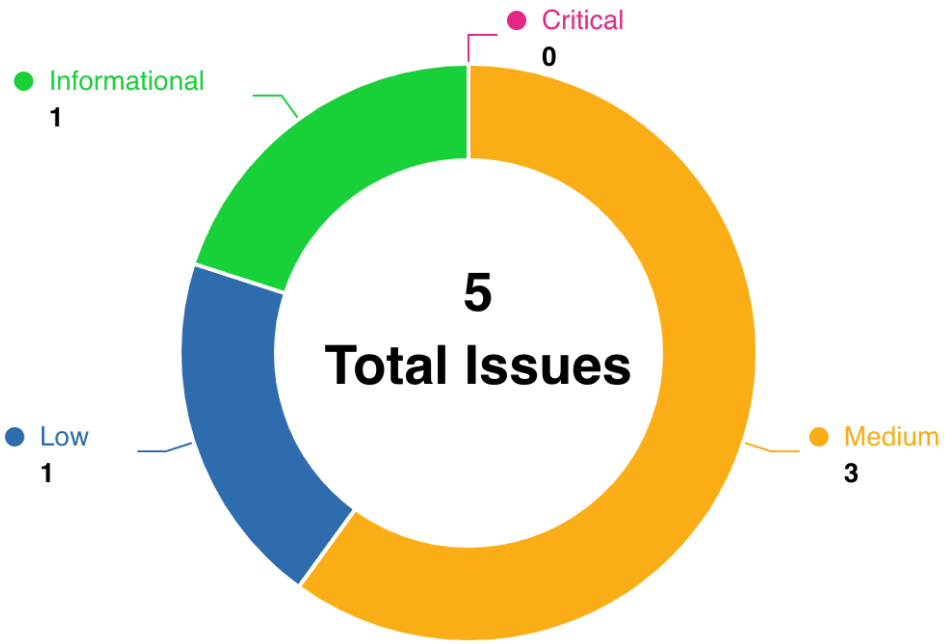
Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PAR-1	Lack view function specified in EIP-2535	Code Style	Informational	Reported	comcat
PAR-2	NTokenMoonBirds may not be able to receive airdrops	Logical	Medium	Reported	thereksfour
PAR-3	WETH9 Compatibility issues in `PoolCore` contract `supplyWithPermit` function	Logical	Low	Reported	w2ning
PAR-4	`WETHGateway.repayETH` will fail when `msg.value > paybackAmount` due to incorrect parameter setting	Logical	Medium	Reported	thereksfour
PAR-5	`WPunkGateway` functions should declare `payable` to buy punks	Logical	Medium	Reported	thereksfour

PAR-1:Lack view function specified in EIP-2535

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/protocol/libraries/paraspace-upgradeability/ParaProxy.sol#L13	Reported	comcat

Code

```
13:    constructor(address _contractOwner) payable {
```

Description

comcat : The ParaProxy is a custom implementation of EIP-2535, it implements the core concept of diamond proxy, for example the `updateImplementaion` function, as well as the `ImplementationUpdated` event. however, according to EIP-2535, it should also implement the following view function, for the purpose of easy check.

```
function facets() external view returns (Facet[] memory facets_);
function facetFunctionSelectors(address _facet) external view returns (bytes4[] memory
facetFunctionSelectors_);
function facetAddresses() external view returns (address[] memory facetAddresses_);
function facetAddress(bytes4 _functionSelector) external view returns (address facetAddress_);
```

currently, the ProxyStorage is private, and it is hard to check the corresponding facet address and selectors.

Recommendation

comcat : Stick to the EIP-2535 standard, implement those view function.

Client Response

TBD

PAR-2:NTokenMoonBirds may not be able to receive airdrops

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	code/contracts/protocol/tokenization/NTokenMoonBirds.sol#L63-L77 code/contracts/protocol/tokenization/NToken.sol#L136-L149	Reported	thereksfour

Code

```
63:    function onERC721Received(
64:        address operator,
65:        address from,
66:        uint256 id,
67:        bytes memory
68:    ) external virtual override returns (bytes4) {
69:        // only accept MoonBird tokens
70:        require(msg.sender == _underlyingAsset, Errors.OPERATION_NOT_SUPPORTED);
71:
72:        // if the operator is the pool, this means that the pool is transferring the token to
this contract
73:        // which can happen during a normal supplyERC721 pool tx
74:        if (operator == address(POOL)) {
75:            return this.onERC721Received.selector;
76:        }
77:

136:    function rescueERC721(
137:        address token,
138:        address to,
139:        uint256[] calldata ids
140:    ) external override onlyPoolAdmin {
141:        require(
142:            token != _underlyingAsset,
143:            Errors.UNDERLYING_ASSET_CAN_NOT_BE_TRANSFERRED
144:        );
145:        for (uint256 i = 0; i < ids.length; i++) {
146:            IERC721(token).safeTransferFrom(address(this), to, ids[i]);
147:        }
148:        emit RescueERC721(token, to, ids);
149:    }
```

Description

thereksfour : For most NToken, some airdrops that are actively minted to the holder's address can be withdrawn and later distributed by the PoolAdmin calling the rescueERC721 function.

```
    function rescueERC721(
        address token,
        address to,
```

```
        uint256[] calldata ids
    ) external override onlyPoolAdmin {
        require(
            token != _underlyingAsset,
            Errors.UNDERLYING_ASSET_CAN_NOT_BE_TRANSFERRED
        );
        for (uint256 i = 0; i < ids.length; i++) {
            IERC721(token).safeTransferFrom(address(this), to, ids[i]);
        }
        emit RescueERC721(token, to, ids);
    }
```

However, in the onERC721Received function of the NTokenMoonBirds contract, due to the requirement that the sender can only be the MoonBird contract, when safemint()/safetransferfrom() is called to send the airdrop NFTs to the NTokenMoonBirds contract, the transaction will fail, thus preventing NTokenMoonBirds from receiving these airdrops.

```
    function onERC721Received(
        address operator,
        address from,
        uint256 id,
        bytes memory
    ) external virtual override returns (bytes4) {
        // only accept MoonBird tokens
        require(msg.sender == _underlyingAsset, Errors.OPERATION_NOT_SUPPORTED);
```

For example, Moonbirds Oddities are actively minted to the holder's address.
https://etherscan.io/tx/0x3af5de8b6a8c55aac033d57e1b110e8340abf4dcd289ebda889a44f9f9dc613d

Recommendation

thereksfour : Consider allowing the NTokenMoonBirds contract to receive NFTs from other addresses and only call POOL.supportERC721FromNToken when msg.sender == _underlyingAsset

```
    function onERC721Received(
        address operator,
        address from,
        uint256 id,
        bytes memory
    ) external virtual override returns (bytes4) {
        // only accept MoonBird tokens
-        require(msg.sender == _underlyingAsset, Errors.OPERATION_NOT_SUPPORTED);

        // if the operator is the pool, this means that the pool is transferring the token to this
contract
        // which can happen during a normal supplyERC721 pool tx
        if (operator == address(POOL)) {
            return this.onERC721Received.selector;
        }
+     if(msg.sender == _underlyingAsset){
        // supply the received token to the pool and set it as collateral
        DataTypes.ERC721SupplyParams[]
            memory tokenData = new DataTypes.ERC721SupplyParams[](1);

        tokenData[0] = DataTypes.ERC721SupplyParams({
            tokenId: id,
            useAsCollateral: true
        });
```

```
        POOL.supplyERC721FromNToken(_underlyingAsset, tokenData, from);
+ }
        return this.onERC721Received.selector;
    }
```

Client Response

TBD

PAR-3:WETH9 Compatibility issues in `PoolCore` contract `supplyWithPermit` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/pool/PoolCore.sol#L156	Reported	w2ning

Code

```
156:    function supplyWithPermit(
```

Description

w2ning : Same Issue as Multichain router v4 vulnerability.

https://medium.com/multichainorg/action-required-critical-vulnerability-for-six-tokens-6b3cbd22bfc0

WETH9 contract has no `permit` function, But there is a `fallback` function, when you call WETH9 with `permit` function would not be revert。

The impact is that It may cause some unexpected calls to complete successfully

Recommendation

w2ning : Check the token address that does not support the 'permit' function

Consider below fix in the `PoolCore.supplyWithPermit()` function

```
function supplyWithPermit(
    ...
    ) external virtual override nonReentrant {

   require(asset != weth9,"WETH9 does not support Permit");

   ...
}
```

Client Response

TBD

PAR-4: `WETHGateway.repayETH` will fail when `msg.value > paybackAmount` due to incorrect parameter setting

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	code/contracts/ui/WETHGateway.sol#L92-L113 code/contracts/protocol/libraries/logic/BorrowLogic.sol#L188-L194	Reported	thereksfour

Code

```
92:     function repayETH(uint256 amount, address onBehalfOf)
93:         external
94:         payable
95:         override
96:         nonReentrant
97:     {
98:         uint256 variableDebt = Helpers.getUserCurrentDebt(
99:             onBehalfOf,
100:             IPool(pool).getReserveData(address(WETH)).variableDebtTokenAddress
101:         );
102:
103:         uint256 paybackAmount = variableDebt;
104:
105:         if (amount < paybackAmount) {
106:             paybackAmount = amount;
107:         }
108:         require(
109:             msg.value >= paybackAmount,
110:             "msg.value is less than repayment amount"
111:         );
112:         WETH.deposit{value: paybackAmount}();
113:         IPool(pool).repay(address(WETH), msg.value, onBehalfOf);

188:         } else {
189:             // send paybackAmount from user to reserve
190:             IERC20(params.asset).safeTransferFrom(
191:                 msg.sender,
192:                 reserveCache.xTokenAddress,
193:                 paybackAmount
194:             );
```

Description

thereksfour : In `WETHGateway.repayETH`, if `msg.value > paybackAmount`, since the amount parameter of `pool.repay` is `msg.value` instead of `paybackAmount`, `BorrowLogic.executeRepay` will fail due to insufficient WETH amount.

Consider user A has a debt of 5 ETH, user A calls `WETHGateway.repayETH` to repay the debt, where `amount` = 2 ETH, `msg.value` = 3 ETH.

Since `amount` = 2 ETH, the `paybackAmount` is also 2 ETH, and exchanged for 2 WETH, the excess 1 ETH will be refunded, and when calling `pool.repay`, `amount` = `msg.value` = 3 ETH.

In the `BorrowLogic.executeRepay` function, `paybackAmount` = 3 ETH, and try to transfer 3 WETH from WETHGateway to the pool, because only 2 WETH was exchanged before, this step will fail.

Note: If there are WETHs accidentally sent by users in the contract, malicious users will be able to use these WETHs to repay debts.

Recommendation

thereksfour : Change to

```
        WETH.deposit{value: paybackAmount}();
-       IPool(pool).repay(address(WETH), msg.value, onBehalfOf);
+      IPool(pool).repay(address(WETH), paybackAmount, onBehalfOf);
```

Client Response

TBD

PAR-5: `WPunkGateway` functions should declare `payable` to buy punks

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/ui/WPunkGateway.sol#L77-L95 • code/contracts/ui/WPunkGateway.sol#L129-L155 • code/contracts/ui/WPunkGateway.sol#L167-L193	Reported	thereksfour

Code

```
77:     function supplyPunk(
78:         DataTypes.ERC721SupplyParams[] calldata punkIndexes,
79:         address onBehalfOf,
80:         uint16 referralCode
81:     ) external nonReentrant {
82:         for (uint256 i = 0; i < punkIndexes.length; i++) {
83:             Punk.buyPunk(punkIndexes[i].tokenId);
84:             Punk.transferPunk(proxy, punkIndexes[i].tokenId);
85:             // gatewayProxy is the sender of this function, not the original gateway
86:             WPunk.mint(punkIndexes[i].tokenId);
87:         }
88:
89:         Pool.supplyERC721(
90:             address(WPunk),
91:             punkIndexes,
92:             onBehalfOf,
93:             referralCode
94:         );
95:     }

129:     function acceptBidWithCredit(
130:         bytes32 marketplaceId,
131:         bytes calldata payload,
132:         DataTypes.Credit calldata credit,
133:         uint256[] calldata punkIndexes,
134:         uint16 referralCode
135:     ) external nonReentrant {
136:         for (uint256 i = 0; i < punkIndexes.length; i++) {
137:             Punk.buyPunk(punkIndexes[i]);
138:             Punk.transferPunk(proxy, punkIndexes[i]);
139:             // gatewayProxy is the sender of this function, not the original gateway
140:             WPunk.mint(punkIndexes[i]);
141:
142:             IERC721(wpunk).safeTransferFrom(
143:                 address(this),
144:                 msg.sender,
```

```
145:                 punkIndexes[i]
146:             );
147:         }
148:         Pool.acceptBidWithCredit(
149:             marketplaceId,
150:             payload,
151:             credit,
152:             msg.sender,
153:             referralCode
154:         );
155:     }

167:     function batchAcceptBidWithCredit(
168:         bytes32[] calldata marketplaceIds,
169:         bytes[] calldata payloads,
170:         DataTypes.Credit[] calldata credits,
171:         uint256[] calldata punkIndexes,
172:         uint16 referralCode
173:     ) external nonReentrant {
174:         for (uint256 i = 0; i < punkIndexes.length; i++) {
175:             Punk.buyPunk(punkIndexes[i]);
176:             Punk.transferPunk(proxy, punkIndexes[i]);
177:             // gatewayProxy is the sender of this function, not the original gateway
178:             WPunk.mint(punkIndexes[i]);
179:
180:             IERC721(wpunk).safeTransferFrom(
181:                 address(this),
182:                 msg.sender,
183:                 punkIndexes[i]
184:             );
185:         }
186:         Pool.batchAcceptBidWithCredit(
187:             marketplaceIds,
188:             payloads,
189:             credits,
190:             msg.sender,
191:             referralCode
192:         );
193:     }
```

Description

thereksfour : `WPunkGateway.supplyPunk`, `WPunkGateway.acceptBidWithCredit` and `WPunkGatewaybatchAcceptBidWithCredit` should declare `payable` and send Ether when calling `Punk.buyPunk`.

Recommendation

thereksfour : Consider adding the payable attribute to WPunkGateway.providePunk/acceptBidWithCredit/batchAcceptBidWithCredit and sending ETH when calling Punk.buyPunk in WPunkGateway to allow users to buy punk directly from the sale.

```
    function supplyPunk(
        DataTypes.ERC721SupplyParams[] calldata punkIndexes,
        address onBehalfOf,
        uint16 referralCode
```

```
-     ) external nonReentrant {
+     ) external payable nonReentrant {
        for (uint256 i = 0; i < punkIndexes.length; i++) {
-             Punk.buyPunk(punkIndexes[i].tokenId);
+             Punk.buyPunk{value:msg.value}(punkIndexes[i].tokenId);
```

Client Response

TBD

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.