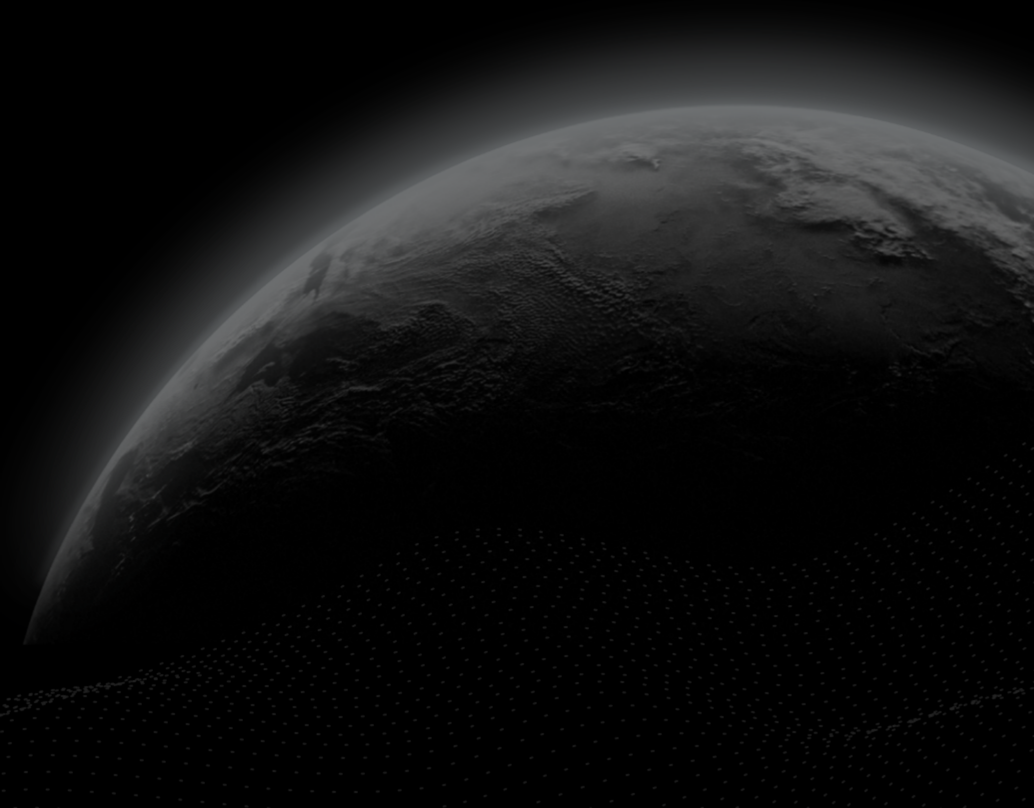
CERTIK

Security Assessment

ParaSpace - NFT Money Market

CertiK Verified on Oct 25th, 2022



CertiK Verified on Oct 25th, 2022

ParaSpace - NFT Money Market

The security assessment was prepared by CertiK, the leader in Web3.0 security.

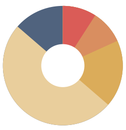
Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending, NFT	BSC	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 10/25/2022	N/A

CODEBASE
https://github.com/para-space/paraspace-core
...View All

COMMITS
aec6ed0ddda43ad3cfbd359c9ffd0d82f45ed6d7
...View All

Vulnerability Summary



22	14	0	2	6	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

2	Critical	2 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	1 Resolved, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
4	Medium	3 Resolved, 1 Partially Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
11	Minor	7 Resolved, 1 Partially Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
3	Informational	1 Resolved, 2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | PARASPACE - NFT MONEY MARKET

CODEBASE | PARASPACE - NFT MONEY MARKET

Repository

https://github.com/para-space/paraspace-core

Commit

aec6ed0ddda43ad3cfbd359c9ffd0d82f45ed6d7

AUDIT SCOPE | PARASPACE - NFT MONEY MARKET

30 files audited ● 13 files with Acknowledged findings ● 17 files without findings

ID	File	SHA256 Checksum
● ACL	contracts/protocol/configuration/ACLManager.sol	95cef06ac33289cadab6d5793999801cd026a2ff44da116f9fa03d21417e28a7
● PAP	contracts/protocol/configuration/PoolAddressesProvider.sol	9450d6851d1f24115c05704dfc28e9581720161373951bb85e3276d6f24656fd
● PAR	contracts/protocol/configuration/PoolAddressesProviderRegistry.sol	267f4dc860bd1c09577abec389e4689fd09f62c8f8a7f89001c21bdaf7d61ee5
● POS	contracts/protocol/configuration/PriceOracleSentinel.sol	c00b70381a6300be6c5df7d6c9d83cbe0708bc839071b16234d1fc638d7d6b29
● RCB	contracts/protocol/libraries/configuration/ReserveConfiguration.sol	aaed53612178e091e8bffad3dfa97a74dbe15006c587db5bcd29961ba7b24b5a
● BLB	contracts/protocol/libraries/logic/BorrowLogic.sol	003c34b961155cc4e8a9cd068441dfa10c7da3a971c7f8f29aacef8cd25c97f0
● LLB	contracts/protocol/libraries/logic/LiquidationLogic.sol	3e8486b0660831aa75c34206921c71544bd70ea9c06adfd41df2253687951af9
● MLB	contracts/protocol/libraries/logic/MarketplaceLogic.sol	c693104f7cae331d7ff314980f7d005ba2fa0eb4738a8f05619e2037dc663af9
● POO	contracts/protocol/pool/Pool.sol	3807c300c1e494c0a0d985f2055f9ef96722dcd7d0d740abac209c95100dc828
● PCB	contracts/protocol/pool/PoolConfigurator.sol	9cbb57a2f63c9466b02f6e1ab622274e8207910ffd69c11cb13c744c6f7275e5
● MIE	contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol	f28e7c94e4402a959239b08c6a4701b022ab5075ecacbba1343067dfc46accb0
● NTB	contracts/protocol/tokenization/NToken.sol	746b6de828f7c5cb8b3151da30cdb9b34630737e5e0b375e50177b0aab1a84a6
● NTU	contracts/protocol/tokenization/NTokenUniswapV3.sol	ec607ab384bd7bd4b5c16d34f67c58aab4911dc8afdba14d458addf26f4b3e1c

ID	File	SHA256 Checksum
UCB	contracts/protocol/libraries/configuration/UserConfiguration.sol	86aaf1f476e75a6bc50a08347335c979216a7ff3645045529141227eb083943e
ERR	contracts/protocol/libraries/helpers/Errors.sol	a742573fa626a858b7829c1476f0079df6367c31bb0e75c3c8bc6682ac6950b2
CLB	contracts/protocol/libraries/logic/ConfiguratorLogic.sol	68a6df4db00045eb06e97c03ef7c85d708a6f89be0620a8e428ed06bbaf221ab
GLB	contracts/protocol/libraries/logic/GenericLogic.sol	d925aae0a3678752b6c1ae8f5b64792dc44d6710de955b3ecf16ded62e7958cb
PLB	contracts/protocol/libraries/logic/PoolLogic.sol	fbc0fa3c120e1139743c7ce837dfab052d1c806b9fcd683b39bb843def0f2df9
RLB	contracts/protocol/libraries/logic/ReserveLogic.sol	703723bcc77a89ef2f119f3be60a61b2b6c02650be40dce1283ec14a7b4b82d6
SLB	contracts/protocol/libraries/logic/SupplyLogic.sol	bb14d9108627998edb59cef3468329c79a10734faf459d4e107c7d90b48e7a16
VLB	contracts/protocol/libraries/logic/ValidationLogic.sol	229d1f71713b051ca70d6fc24436095280b64325d0bb6c404673a5d168cc6faa
MUB	contracts/protocol/libraries/math/MathUtils.sol	d30ce03102a94e569418949d72632e1dc9cab8913812c3ae9943e6c8843f0d36
PMB	contracts/protocol/libraries/math/PercentageMath.sol	7fe9afd04a2494c9c257ab118ceceb27325b457651a14b5c0061ebd83bcb8fc6
WRM	contracts/protocol/libraries/math/WadRayMath.sol	b9009088a40469b39b5dca345c2da9d861a5f5cd818724aed7566ba6085c17d9
CIT	contracts/protocol/libraries/types/ConfiguratorInputTypes.sol	cce746f852074294e3640e89f8561164e969eda2fccecd85c4560a64fe36de1a
DTB	contracts/protocol/libraries/types/DataTypes.sol	6e891e3ee4dc4b0f5a4bf747dce5c34096517539face1ed24992151be98348f9
PSB	contracts/protocol/pool/PoolStorage.sol	2dfb1735a4e8ca9d81e8a7fce2af4a9dd8de88c40a495706af9f26b516873455
DRI	contracts/protocol/pool/DefaultReserveInterestRateStrategy.sol	a01e7cab83a7ffcda8237ca31caca0b629c315a32fdde1359d2cb455ea2e0819

ID	File	SHA256 Checksum
SBT	contracts/protocol/tokenization/base/ScaledBalanceTokenBaseERC721.sol	7eb6417815fcca7cdde5e6991d26aa61b498c5a462054719f3cd7f6428769ce0
NTM	contracts/protocol/tokenization/NTokenMoonBirds.sol	c8e983646c3ab6ad73739d6b6f20214113ba2549466d5aed63ab986c67e1f76c

APPROACH & METHODS | PARASPACE - NFT MONEY MARKET

This report has been prepared for ParaSpace to discover issues and vulnerabilities in the source code of the ParaSpace - NFT Money Market project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

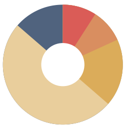
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | PARASPACE - NFT MONEY MARKET



22	2	2	4	11	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for ParaSpace - NFT Money Market. Through this audit, we have uncovered 22 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ParaSpace-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
GLB-01	Potential Incorrect Price Risk	Volatile Code	Minor	Resolved
MLB-01	Potential Financial Loss	Logical Issue	Medium	Resolved
NTB-01	`NToken.permit()` Doesn't Check The Token Owner	Logical Issue	Critical	Resolved
NTH-01	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
NTU-01	Incompatibility With Deflationary Tokens	Logical Issue	Medium	Resolved
NTU-02	Incorrect Conditional Statement	Logical Issue	Minor	Resolved
PCB-01	Lack Of Reasonable Boundary	Volatile Code	Minor	Partially Resolved
POL-01	Functions Not Restricted	Logical Issue	Medium	Resolved
POL-02	Lack Of Account Validation	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
POO-01	Potential Flashloan Attack	Logical Issue	Critical	Resolved
POO-02	Potential Reentrancy Attack	Logical Issue	Major	Resolved
PRO-01	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
PRO-02	Unused Return Value	Volatile Code	Minor	Acknowledged
PRO-03	`initialize()` Is Unprotected	Volatile Code	Minor	Acknowledged
PRT-01	Check-Effects-Interact Pattern Not Implemented	Volatile Code	Medium	Partially Resolved
SLB-01	Redundant `else` Clause	Logical Issue	Minor	Resolved
TOK-01	Missing Zero Address Validation	Volatile Code	Minor	Resolved
VLB-01	Redundant Code	Volatile Code	Minor	Resolved
ParaSpace-02	Potential Risks Of Pool Establishment	Control Flow	Informational	Acknowledged
POO-03	Discussion On Borrow With Credit	Control Flow	Informational	Resolved
PRO-05	Incorrect Comments	Inconsistency	Informational	Acknowledged

PARASPACE-01 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor		Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party `UniswapV3` , `OpenSea` , `X2Y2` , `MoonBird` and NFT Oracle protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

The `ParaSpace` protocol allows users to borrow assets using NFT as collateral. If NFT prices fluctuate significantly in the third-party markets, the Supplier's health factory may fluctuate as well. This is a potential risk to this protocol and to the Supplier.

Recommendation

We understand that the business logic of `ParaSpace` requires interaction with `uniswapV3` , `openSea` , etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`[ParaSpace]` : No action needed, we consider it to be safe.

GLB-01 | POTENTIAL INCORRECT PRICE RISK

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/libraries/logic/GenericLogic.sol (base): 356~360	Resolved

Description

- GenericLogic.sol contract `_getUserBalanceInBaseCurrency()` method

```
uint256 assetPrice;

......

if (INToken(xTokenAddress).getAtomicPricingConfig()) {
  uint256 totalBalance = INToken(xTokenAddress).balanceOf(user);

  for (uint256 index = 0; index < totalBalance; index++) {
    uint256 tokenId = IERC721Enumerable(xTokenAddress)
      .tokenOfOwnerByIndex(user, index);
    if (
      ICollaterizableERC721(xTokenAddress).isUsedAsCollateral(
        tokenId
      )
    ) {
      // TODO use getTokensPrices instead if it saves gas
      assetPrice = IPriceOracleGetter(oracle).getTokenPrice(
        currentReserveAddress,
        tokenId
      );
      balance += assetPrice;
    }
  }
}

......

unchecked {
  return (balance / assetUnit, assetPrice);
}
```

According to the above statement, the method `_getUserBalanceInBaseCurrency()` will return the `assetPrice` of the last item.

And referring to the client's technical documentation, `atomic pricing` is defined as:

the first notable difference between typical floor-based and UniV3 is that each token has a different price based on the ERC20 token composition inside the LP token. This means that each UniV3 token has a different price, and in the statement logic, the local variable `assetPrice` will be overwritten by the value of the last item in the loop.

We understand that for NFT assets, the user cannot borrow them now. However, the result returned by this method is incorrect and might not make sense.

Recommendation

We recommend the client to make sure that the code here can match the design intent.

Alleviation

The ParaSpace team resolved this issue in commit `aec6ed0ddda43ad3cfbd359c9ffd0d82f45ed6d7`.

MLB-01 | POTENTIAL FINANCIAL LOSS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/protocol/libraries/logic/MarketplaceLogic.sol (base): 279~291	Resolved

Description

```
279    if (reserve.xTokenAddress == address(0)) {
280        address underlyingAsset = INToken(token)
281            .UNDERLYING_ASSET_ADDRESS();
282        reserve = reservesData[underlyingAsset];
283        bool isNToken = reserve.xTokenAddress == token;
284
285        require(isNToken, Errors.ASSET_NOT_LISTED);
286        if (!userConfig.isUsingAsCollateral(reserve.id)) {
287            userConfig.setUsingAsCollateral(reserve.id, true);
288        }
289        // No need to supply anymore because it's already NToken
290        continue;
291    }
```

According to the `if` condition `reserve.xTokenAddress == address(0)` of the above statement from method `_repay()`, this means that the purchased token is an `NToken`. However, this token will be locked in the `pool` of the contract forever. This may be incorrect.

Recommendation

We recommend the client ensuring the logical correctness.

Alleviation

ParaSpace modified the related code in commit `5139c7bc36884b7337eb48dc2e39372f6688786b`, the protocol will always use `pool` as NFT purchase recipient.

NTB-01 | `NToken.permit()` DOESN'T CHECK THE TOKEN OWNER

Category	Severity	Location	Status
Logical Issue	● Critical	contracts/protocol/tokenization/NToken.sol: 288~291	● Resolved

Description

```
require(owner == ecrecover(digest, v, r, s), Errors.INVALID_SIGNATURE);
_nonces[owner] = currentValidNonce + 1;
_approve(spender, value);
```

`NToken.permit()` allows the `spender` to transfer in the future the token with id `value` if correctly signed by `owner` message provided. However, it is not checked that `_isApprovedOrOwner(owner, value)` . As a result, anyone can get approval for any token.

The `value` argument name is misleading.

Recommendation

We recommend

1. Rename the `value` argument to `tokenId` .
2. Omitting the `owner` argument. Setting `address owner = ownerOf(tokenId)` .
3. Checking `require(signer == owner || isApprovedForAll(owner, signer))` .
4. Renaming the `PERMIT_TYPEHASH` and changing it correspondingly.

Alleviation

ParaSpace team removed the `permit()` function in commit `636e92a9d5d2b5a4cc659e1b1e0c5942b84ee7e6`

NTH-01 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/tokenization/NToken.sol (base): 155	Resolved

Description

The return value of the transfer()/transferFrom() call is not checked.

```
155        IERC20(token).transfer(to, amount);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

The team heeded our advice and resolved this issue in commit `636e92a9d5d2b5a4cc659e1b1e0c5942b84ee7e6`.

NTU-01 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/protocol/tokenization/NTokenUniswapV3.sol: 200, 203, 229, 234, 264~270	Resolved

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived at the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

```
200             IERC20(token0).safeTransferFrom(sender, address(this), amountAdd0);
```

- Transferring tokens by `amountAdd0`.

```
228             uint256 refund0 = amountAdd0 - amount0;
```

- The `amountAdd0` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
203             IERC20(token1).safeTransferFrom(sender, address(this), amountAdd1);
```

- Transferring tokens by `amountAdd1`.

```
233             uint256 refund1 = amountAdd1 - amount1;
```

- The `amountAdd1` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
        _increaseLiquidityCurrentRange(
            tokenId,
            amountAdd0,
            amountAdd1,
            amount0Min,
            amount1Min
        );
```

- Transferring tokens by `amountAdd0`.
- This function call executes the following operation.
- In `NTokenUniswapV3._increaseLiquidityCurrentRange`,
    - `IERC20(token0).safeTransferFrom(sender, address(this), amountAdd0);`

```
        _increaseLiquidityCurrentRange(
            tokenId,
            amountAdd0,
            amountAdd1,
            amount0Min,
            amount1Min
        );
```

- This function call executes the following operation.
- In `NTokenUniswapV3._increaseLiquidityCurrentRange`,
    - `uint256 refund0 = amountAdd0 - amount0;`
- The `amountAdd0` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

The client removed this code in commit 9be10233cd58c73e48df45f3f91538ee72885c6f.

NTU-02 | INCORRECT CONDITIONAL STATEMENT

Category	Severity	Location	Status
Logical Issue	Minor	contracts/protocol/tokenization/NTokenUniswapV3.sol: 49~51	Resolved

Description

Referring to the comments, the logic may occur in a normal supplyERC721 pool transaction. However, when the operator is `POOL` the transaction will be reverted. That means the `UniswapV3` tokens cannot be supplied in this protocol.

```
47 // if the operator is the pool, this means that the pool is transferring the
token to this contract
48 // which can happen during a normal supplyERC721 pool tx
49 if (operator == address(POOL)) {
50     revert(Errors.OPERATION_NOT_SUPPORTED);
51 }
```

Recommendation

We recommend reviewing the logic to ensure it meets the design intent.

Alleviation

`[ParaSpace]` : Yes, it's expected because for `Moonbirds` & `Uniswap` , users will need to transfer to `NToken` then `NToken` will supply for them. And `UniswapV3Gateway` does the `POOL.supplyERC721FromNToken` step.

- UniswapV3Gateway.sol

```
function supplyUniswapV3(
  address pool,
  DataTypes.ERC721SupplyParams[] calldata tokenIds,
  address onBehalfOf
) external {
  for (uint256 index = 0; index < tokenIds.length; index++) {
    IERC721(UNISWAP_V3_POSITION_MANAGER).safeTransferFrom(
      msg.sender,
      address(
        POOL
          .getReserveData(UNISWAP_V3_POSITION_MANAGER)
          .xTokenAddress
      ),
      tokenIds[index].tokenId
    );
  }

  POOL.supplyERC721FromNToken(
    UNISWAP_V3_POSITION_MANAGER,
    tokenIds,
    onBehalfOf
  );
}
```

PCB-01 | LACK OF REASONABLE BOUNDARY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/pool/PoolConfigurator.sol: 195, 274, 313, 327 ~337, 341	Partially Resolved

Description

The variables `auctionRecoveryHealthFactor` , `newReserveFactor` , `newBorrowCap` , and `newFee` do not have reasonable boundaries, so they can be given arbitrary values after deploying.

Recommendation

We recommend adding reasonable upper and lower boundaries to all the configuration variables.

Alleviation

The team heeded our advice and added a validation to the auction recovery health factor in commit `355402a64a9c857d9c13b46d16bf813a3186fd56` .

POL-01 | FUNCTIONS NOT RESTRICTED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/protocol/pool/Pool.sol (base): 358, 393	Resolved

Description

```
function batchBuyWithCredit(
      bytes32[] calldata marketplaceIds,
      bytes[] calldata payloads,
      DataTypes.Credit[] calldata credits,
      address onBehalfOf,
      uint16 referralCode
  ) external payable virtual override nonReentrant {
```

```
function buyWithCredit(
    bytes32 marketplaceId,
    bytes calldata payload,
    DataTypes.Credit calldata credit,
    address onBehalfOf,
    uint16 referralCode
  ) external payable virtual override nonReentrant {
```

Both of the `buyWithCredit()` and `batchBuyWithCredit()` are external functions without any validation. Any user can call them with any parameter. At the same time, these functions do not validate the caller's allowance from the `onBehalfOf` account. Not verifying that the caller has allowance from the `onBehalfOf` account could be a vulnerability.

Recommendation

Given the significant risk associated with these two methods, we recommended verifying order signatures and credit signatures.

Alleviation

`[ParaSpace]`: To address this issue, we will remove `onBahalfOf` parameter and remove `buyWithCredit` call from `WETHGateway`. In this case, we allow only a user to buy NFT using their own credit. The related pull request is #71.

POL-02 | LACK OF ACCOUNT VALIDATION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/protocol/pool/Pool.sol (base): 149	Resolved

Description

```
function supplyERC721FromNToken(
  address asset,
  DataTypes.ERC721SupplyParams[] calldata tokenData,
  address onBehalfOf
) external virtual override nonReentrant {
  SupplyLogic.executeSupplyERC721FromNToken(
    _reserves,
    _usersConfig[onBehalfOf],
    DataTypes.ExecuteSupplyERC721Params({
      asset: asset,
      tokenData: tokenData,
      onBehalfOf: onBehalfOf,
      actualSpender: address(0),
      referralCode: 0
    })
  );
}
```

In the function `Pool.supplyERC721FromNToken()` , any user can call it to supply ERC721 from other accounts without restriction.

Recommendation

We recommend adding checks to ensure that only the NToken contract is allowed to call this function.

Alleviation

The team heeded our advice and resolved this issue in commit `204812009a0240b5ce41c96067e5c4fdaaa03774` .

POO-01 | POTENTIAL FLASHLOAN ATTACK

Category	Severity	Location	Status
Logical Issue	● Critical	contracts/protocol/pool/Pool.sol: 292~293	● Resolved

Description

The following check is performed when borrowing if the interest rate mode is stable:

```
require(
    !params.userConfig.isUsingAsCollateral(
        reservesData[params.asset].id
    ) ||
        params.reserveCache.reserveConfiguration.getLtv() == 0 ||
        params.amount >
        IToken(params.reserveCache.xTokenAddress).balanceOf(
            params.userAddress
        ),
    Errors.COLLATERAL_SAME_AS_BORROWING_CURRENCY
);
```

So the current attack only applies if the interest rate mode is variable. The following slightly tweaked attack works in either interest rate mode.

Attack Flow:

1. An exploiter creates two contracts A and B.
2. Contract A flash loans 100 ETH from other protocols, it then supplies 100 ETH and is minted 100 ETH of PToken (assuming the asset is ETH), they then borrow 80% of the locked ETH in some other Token. (assuming up to 80% can be borrowed).
3. Contract A transfers the borrowed Token's to Contract B, which then swaps them for around 80 ETH.
4. Contract B then supplies 80 ETH and is minted 80 ETH worth of PToken. Subsequently, they borrow 64 ETH worth of some other Token, which they then swap for around 64 ETH.
5. Contract B uses its own 80 ETH worth of PToken to pay off Contract A's debt.
6. Contract A withdraws all 100 ETH supplied by burning its 100 ETH of PToken, as its debt is now cleared.
7. Contract A repays the 100 ETH flash loan.
8. The exploiter gains around 64 ETH in profit as they still have the 64 ETH in contract B. (Not accounting for the swap and flash loan fees.)

Recommendation

We recommend only allowing the `msg.sender` to repay their own debts using PToken.

Alleviation

The team heeded the recommendation and resolved the finding in commit b0666aa533fa470adacdb24094c2583c04adf7be005f68fe7dc37c707bea50ab

POO-02 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Logical Issue	Major	contracts/protocol/pool/Pool.sol: 527~531, 571~578, 598~602, 618~622	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects.

If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

For example, the hacker can call the `withdrawERC721()` method and use the hook `_checkOnERC721Received` method of the ERC721 receiver to reenter the method `liquidationERC721()` .

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The team heeded the recommendation and resolved the finding in commit 003de864bf36010729cf1ee887ba4047b74f8d88.

PRO-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/protocol/configuration/ACLManager.sol: 40~43; contracts/protocol/configuration/PoolAddressesProvider.sol: 54~57, 68~71, 79~82, 101, 113~116, 134~137, 150, 162, 174~177, 216~219, 227, 234~240; contracts/protocol/configuration/PoolAddressesProviderRegistry.sol: 46~50, 72~75; contracts/protocol/configuration/PriceOracleSentinel.sol: 91~93, 100~102; contracts/protocol/pool/Pool.sol: 835~843, 866~870, 876~879, 889~892, 902~905, 916~919, 929~932, 993~997, 1052~1056; contracts/protocol/pool/PoolConfigurator.sol: 84~86, 94, 100~102, 107~109, 114~116, 121~124, 140~145, 192~196, 215~218, 234~237, 248~251, 261~264, 274~277, 292~295, 313~316, 327~330, 341~344, 359~362, 377~380, 396, 407~410	Acknowledged

Description

In the contract `Pool` , the role `onlyPoolAdmin` has authority over the following functions:

- function rescueTokens(), to transfer any ERC20 tokens in the contract to any `to` address.

Any compromise to the `onlyPoolAdmin` account may allow a hacker to take advantage of this authority.

In the contract `Pool` , the role `onlyPoolConfigurator` has authority over the following functions:

- function initReserve(), to initialize a reserve, activate it, assign an `NToken` / `PToken` and debt tokens and an interest rate strategy.
- function dropReserve(), to drop a reserve.
- function setReserveInterestRateStrategyAddress(), to update the address of the interest rate strategy contract.
- function setReserveAuctionStrategyAddress(), to update the address of the auction strategy contract.
- function setReserveDynamicConfigsStrategyAddress(), to update the address of the dynamic configs strategy contract.
- function setConfiguration(), to set the configuration bitmap of the reserve as a whole.
- function setAuctionConfiguration(), to set the auction configuration bitmap of the reserve as a whole.
- function setMaxAtomicTokensAllowed(), to set the maximum allowed atomic tokens per user.

Any compromise to the `onlyPoolConfigurator` account may allow a hacker to take advantage of this authority.

In the contract `ACLManager` , the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- function setRoleAdmin(), to set the role as admin of a specific role.

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority.

In the contract `PoolAddressesProvider` , the role `Owner` has authority over the following functions:

- function setMarketId(), to associate an id with a specific PoolAddressesProvider.
- function setAddress(), to set an address for an id replacing the address saved in the addresses map.
- function setAddressAsProxy(), to update the implementation of a proxy registered with certain `id` . If there is no proxy registered, it will instantiate one and set as implementation the `newImplementationAddress` .
- function setPoolImpl(), to update the implementation of the Pool or creates a proxy setting for the new `pool` implementation when the function is called for the first time.
- function setPriceOracle(), to update the address of the price oracle.
- function setACLManager(), to update the address of the ACL manager.
- function setACLAdmin(), to update the address of the ACL admin.
- function setPriceOracleSentinel(), to update the address of the price oracle sentinel.
- function setPoolDataProvider(), to update the address of the data provider.
- function setWETH(), to update the address of the WETH.
- function setMarketplace(),to update the info of the marketplace.

Any compromise to the `Owner` account may allow a hacker to take advantage of this authority.

In the contract `PoolAddressesProviderRegistry` , the role `Owner` has authority over the following functions:

- function registerAddressesProvider(), to register an addresses provider.
- function unregisterAddressesProvider(), to remove an addresses provider from the list of registered addresses providers.

Any compromise to the `Owner` account may allow a hacker to take advantage of this authority.

In the contract `PriceOracleSentinel` , the role `onlyPoolAdmin` has authority over the following functions:

- function setSequencerOracle(), to update the address of the sequencer oracle.

Any compromise to the `onlyPoolAdmin` account may allow a hacker to take advantage of this authority.

In the contract `PriceOracleSentinel` , the role `onlyRiskOrPoolAdmins` has authority over the following functions:

- function setGracePeriod(), to update the duration of the grace period.

Any compromise to the `onlyRiskOrPoolAdmins` account may allow a hacker to take advantage of this authority.

In the contract `PoolConfigurator` , the role `onlyPoolAdmin` has authority over the following functions:

- function dropReserve(), to drop a reserve entirely.

- function updatePToken(), to update the PToken implementation for the reserve.
- function updateStableDebtToken(), to update the stable debt token implementation for the reserve.
- function updateVariableDebtToken(), to update the variable debt token implementation for the asset.
- function setReserveActive(), to activate or deactivate a reserve.

Any compromise to the `onlyPoolAdmin` account may allow a hacker to take advantage of this authority.

In the contract `PoolConfigurator`, the role `onlyRiskOrPoolAdmins` has authority over the following functions:

- function setReserveBorrowing(), to configure borrowing on a reserve.
- function configureReserveAsCollateral(), to configure the reserve collateralization parameters.
- function configureReserveAsAuctionCollateral(), to configure the reserve collateralization parameters.
- function setReserveStableRateBorrowing(), to enable or disable stable rate borrowing on a reserve.
- function setReserveFreeze(), to freeze or unfreeze a reserve. A frozen reserve doesn't allow any new supply, borrow or rate swap but allows repayments, liquidations, rate rebalances, and withdrawals.
- function setReserveFactor(), to update the reserve factor of a reserve.
- function setSiloedBorrowing(), to set siloed borrowing for an asset.
- function setBorrowCap(), to update the borrow cap of a reserve.
- function setSupplyCap(), to update the supply cap of a reserve.
- function setLiquidationProtocolFee(), to update the liquidation protocol fee of reserve.
- function setReserveInterestRateStrategyAddress(), to set the interest rate strategy of a reserve.
- function setReserveDynamicConfigsStrategyAddress(), to set the dynamic configs strategy of a reserve.
- function setMaxAtomicTokensAllowed(), to set the maximum allowed atomic tokens per user.

Any compromise to the `onlyRiskOrPoolAdmins` account may allow a hacker to take advantage of this authority.

In the contract `PoolConfigurator`, the role `onlyEmergencyOrPoolAdmin` has authority over the following functions:

- function setReservePause(), to pause a reserve. A paused reserve does not allow any interaction (supply, borrow, repay,
    - swap interest rate, liquidate, NToken/PToken transfers).

Any compromise to the `onlyEmergencyOrPoolAdmin` account may allow a hacker to take advantage of this authority.

In the contract `PoolConfigurator`, the role `onlyEmergencyAdmin` has authority over the following functions:

- function setPoolPause(), to pause or unpause all the protocol reserves. In the paused state all the protocol interactions are suspended.

Any compromise to the `onlyEmergencyAdmin` account may allow a hacker to take advantage of this authority.

In the contract `PoolConfigurator`, the role `onlyAssetListingOrPoolAdmins` has authority over the following functions:

- function initReserves(), to initialize multiple reserves.

Any compromise to the `onlyAssetListingOrPoolAdmins` account may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (2/3, 3/5) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

ParaSpace team acknowledged this finding.

PRO-02 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/libraries/logic/BorrowLogic.sol: 290; contracts/protocol/libraries/logic/LiquidationLogic.sol: 651; contracts/protocol/libraries/logic/MarketplaceLogic.sol: 80~88, 134~141, 200, 201; contracts/protocol/pool/Pool.sol: 353~371; contracts/protocol/tokenization/NToken.sol: 205~209; contracts/protocol/tokenization/NTokenUniswapV3.sol: 107~109; contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol: 614~633	Acknowledged

Description

The return value of an external call is not stored in a local or state variable.

```
290   stableDebtToken.burn(user, stableDebt);
```

```
651   INToken(vars.collateralXToken).burn(params.user, msg.sender, tokenIds);
```

```
80    Address.functionDelegateCall(
81      params.marketplace.adapter,
82      abi.encodeWithSelector(
83        IMarketplace.matchAskWithTakerBid.selector,
84        params.marketplace.marketplace,
85        params.payload,
86        priceEth
87      )
88    );
```

```
134   Address.functionDelegateCall(
135     params.marketplace.adapter,
136     abi.encodeWithSelector(
137       IMarketplace.matchBidWithTakerAsk.selector,
138       params.marketplace.marketplace,
139       params.payload
140     )
141   );
```

```
200   IERC20(token).approve(params.marketplace.operator, 0);
```

```
  IERC20(token).approve(params.marketplace.operator, price);
```

```
MarketplaceLogic.executeBuyWithCredit(
  _reserves,
  _reservesList,
  _usersConfig[onBehalfOf],
  DataTypes.ExecuteMarketplaceParams({
    marketplaceId: marketplaceId,
    payload: payload,
    credit: credit,
    ethLeft: msg.value,
    marketplace: marketplace,
    orderInfo: orderInfo,
    WETH: WETH,
    referralCode: referralCode,
    maxStableRateBorrowSizePercent: _maxStableRateBorrowSizePercent,
    reservesCount: _reservesCount,
    oracle: ADDRESSES_PROVIDER.getPriceOracle(),
    priceOracleSentinel: ADDRESSES_PROVIDER.getPriceOracleSentinel()
  })
);
```

```
Address.functionCall(
  airdropContract,
  airdropParams,
  Errors.CALL_AIRDROP_METHOD_FAILED
);
```

```
  INonfungiblePositionManager(_underlyingAsset).decreaseLiquidity(
    params
  );
```

```
try
  IERC721Receiver(to).onERC721Received(
    _msgSender(),
    from,
    tokenId,
    _data
  )
returns (bytes4 retval) {
  return retval == IERC721Receiver.onERC721Received.selector;
} catch (bytes memory reason) {
  if (reason.length == 0) {
    revert(
      "ERC721: transfer to non ERC721Receiver implementer"
    );
  } else {
    assembly {
      revert(add(32, reason), mload(reason))
    }
  }
}
```

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

ParaSpace team acknowledged this finding.

PRO-03 | `initialize()` IS UNPROTECTED

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/pool/Pool.sol: 100~103; contracts/protocol/tokenization/NToken.sol: 61~70	Acknowledged

Description

The function `initialize()` is `public` and can be called by anyone as long as the contract is deployed.

Recommendation

We recommend adding a `_disableInitializers()` function similar to Openzeppelin's or using `constructor() initializer {}`.

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {}
```

This will prevent the calling of `initialize()` directly on the implementation contract. But the proxy will still be able to `initialize()` its storage variables.

Alleviation

ParaSpace team acknowledged this finding.

PRT-01 | CHECK-EFFECTS-INTERACT PATTERN NOT IMPLEMENTED

Category	Severity	Location	Status
Volatile Code	Medium	contracts/protocol/configuration/PoolAddressesProvider.sol (base): 85, 86, 102, 103, 118~120, 121, 271, 272, 278, 305~308; contracts/protocol/libraries/configuration/UserConfiguration.sol (base): 39, 41, 64, 66; contracts/protocol/libraries/logic/BorrowLogic.sol (base): 99~108, 110~118, 122, 125~130, 203~209, 211~217, 220~225, 290, 292~301, 303, 341~347, 349~356, 358~364, 366~375, 378; contracts/protocol/libraries/logic/LiquidationLogic.sol (base): 199, 201~206, 209, 211, 216~220, 226, 387~396, 401~404, 425, 426~431, 441, 443, 447, 463, 495, 496~501, 504~509, 525, 545~549, 586~590, 615~621, 625~633, 635~642; contracts/protocol/libraries/logic/ReserveLogic.sol (base): 105, 106, 207, 208, 209, 279~282, 309~311, 324~326, 331; contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol (base): 514~518, 520~524, 610, 613, 614	Partially Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

PoolAddressesProvider.sol

External call(s)

```
85        address oldImplementationAddress = _getProxyImplementation(id);
```

- This function call executes the following external call(s).
- In `PoolAddressesProvider._getProxyImplementation` ,
  - `InitializableImmutableAdminUpgradeabilityProxy(payableProxyAddress).implementation()`

```
86        _updateImpl(id, newImplementationAddress);
```

- This function call executes the following external call(s).
- In `PoolAddressesProvider._updateImpl` ,
    - `proxy.initialize(newAddress,params)`
- In `PoolAddressesProvider._updateImpl` ,
    - `proxy.upgradeToAndCall(newAddress,params)`

State variables written after the call(s)

```
86        _updateImpl(id, newImplementationAddress);
```

- This function call executes the following assignment(s).
- In `PoolAddressesProvider._updateImpl` ,
    - `_addresses[id] = proxyAddress = address(proxy)`

PoolAddressesProvider.sol

External call(s)

```
102        address oldPoolImpl = _getProxyImplementation(POOL);
```

- This function call executes the following external call(s).
- In `PoolAddressesProvider._getProxyImplementation` ,
    - `InitializableImmutableAdminUpgradeabilityProxy(payableProxyAddress).implementation()`

```
103        _updateImpl(POOL, newPoolImpl);
```

- This function call executes the following external call(s).
- In `PoolAddressesProvider._updateImpl` ,
    - `proxy.initialize(newAddress,params)`
- In `PoolAddressesProvider._updateImpl` ,
    - `proxy.upgradeToAndCall(newAddress,params)`

State variables written after the call(s)

```
_updateImpl(POOL, newPoolImpl);
```

- This function call executes the following assignment(s).
- In `PoolAddressesProvider._updateImpl` ,
    - `_addresses[id] = proxyAddress = address(proxy)`

PoolAddressesProvider.sol

External call(s)

```
address oldPoolConfiguratorImpl = _getProxyImplementation(
    POOL_CONFIGURATOR
);
```

- This function call executes the following external call(s).
- In `PoolAddressesProvider._getProxyImplementation` ,
    - `InitializableImmutableAdminUpgradeabilityProxy(payableProxyAddress).implementation()`

```
_updateImpl(POOL_CONFIGURATOR, newPoolConfiguratorImpl);
```

- This function call executes the following external call(s).
- In `PoolAddressesProvider._updateImpl` ,
    - `proxy.initialize(newAddress,params)`
- In `PoolAddressesProvider._updateImpl` ,
    - `proxy.upgradeToAndCall(newAddress,params)`

State variables written after the call(s)

```
_updateImpl(POOL_CONFIGURATOR, newPoolConfiguratorImpl);
```

- This function call executes the following assignment(s).

- In `PoolAddressesProvider._updateImpl` ,
    - `_addresses[id] = proxyAddress = address(proxy)`

BorrowLogic.sol

External call(s)

```
    (
        isFirstBorrowing,
        reserveCache.nextTotalStableDebt,
        reserveCache.nextAvgStableBorrowRate
    ) = IStableDebtToken(reserveCache.stableDebtTokenAddress).mint(
        params.user,
        params.onBehalfOf,
        params.amount,
        currentStableRate
    );
```

```
    (
        isFirstBorrowing,
        reserveCache.nextScaledVariableDebt
    ) = IVariableDebtToken(reserveCache.variableDebtTokenAddress).mint(
        params.user,
        params.onBehalfOf,
        params.amount,
        reserveCache.nextVariableBorrowIndex
    );
```

State variables written after the call(s)

```
    reserve.updateInterestRates(
        reserveCache,
        params.asset,
        0,
        params.releaseUnderlying ? params.amount : 0
    );
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates` ,
    - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`

- In `ReserveLogic.updateInterestRates` ,
  - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`
- In `ReserveLogic.updateInterestRates` ,
  - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

```
    reserve.updateInterestRates(
        reserveCache,
        params.asset,
        0,
        params.releaseUnderlying ? params.amount : 0
    );
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates` ,
  - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`
- In `ReserveLogic.updateInterestRates` ,
  - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`
- In `ReserveLogic.updateInterestRates` ,
  - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

```
            userConfig.setBorrowing(reserve.id, true);
```

- This function call executes the following assignment(s).
- In `UserConfiguration.setBorrowing` ,
  - `self.data |= bit`
- In `UserConfiguration.setBorrowing` ,
  - `self.data &= ~ bit`

BorrowLogic.sol

External call(s)

```
        (
            reserveCache.nextTotalStableDebt,
            reserveCache.nextAvgStableBorrowRate
        ) = IStableDebtToken(reserveCache.stableDebtTokenAddress).burn(
            params.onBehalfOf,
            paybackAmount
        );
```

```
        reserveCache.nextScaledVariableDebt = IVariableDebtToken(
            reserveCache.variableDebtTokenAddress
        ).burn(
                params.onBehalfOf,
                paybackAmount,
                reserveCache.nextVariableBorrowIndex
            );
```

State variables written after the call(s)

```
        reserve.updateInterestRates(
            reserveCache,
            params.asset,
            params.usePTokens ? 0 : paybackAmount,
            0
        );
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates`,
  - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
  - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
  - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

```
        reserve.updateInterestRates(
            reserveCache,
            params.asset,
            params.usePTokens ? 0 : paybackAmount,
            0
        );
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

BorrowLogic.sol

External call(s)

```
stableDebtToken.burn(user, stableDebt);
```

```
(
    ,
    reserveCache.nextTotalStableDebt,
    reserveCache.nextAvgStableBorrowRate
) = stableDebtToken.mint(
    user,
    user,
    stableDebt,
    reserve.currentStableBorrowRate
);
```

State variables written after the call(s)

```
reserve.updateInterestRates(reserveCache, asset, 0, 0);
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`

- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

BorrowLogic.sol

External call(s)

```
    (
        reserveCache.nextTotalStableDebt,
        reserveCache.nextAvgStableBorrowRate
    ) = IStableDebtToken(reserveCache.stableDebtTokenAddress).burn(
        msg.sender,
        stableDebt
    );
```

```
    (, reserveCache.nextScaledVariableDebt) = IVariableDebtToken(
        reserveCache.variableDebtTokenAddress
    ).mint(
            msg.sender,
            msg.sender,
            stableDebt,
            reserveCache.nextVariableBorrowIndex
        );
```

```
    reserveCache.nextScaledVariableDebt = IVariableDebtToken(
        reserveCache.variableDebtTokenAddress
    ).burn(
            msg.sender,
            variableDebt,
            reserveCache.nextVariableBorrowIndex
        );
```

```
    (
        ,
        reserveCache.nextTotalStableDebt,
        reserveCache.nextAvgStableBorrowRate
    ) = IStableDebtToken(reserveCache.stableDebtTokenAddress).mint(
        msg.sender,
        msg.sender,
        variableDebt,
        reserve.currentStableBorrowRate
    );
```

State variables written after the call(s)

```
378         reserve.updateInterestRates(reserveCache, asset, 0, 0);
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates`,
  - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
  - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
  - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

LiquidationLogic.sol

External call(s)

```
199         _burnDebtTokens(params, vars);
```

- This function call executes the following external call(s).
- In `LiquidationLogic._burnDebtTokens`,
  - `vars.debtReserveCache.nextScaledVariableDebt = IVariableDebtToken(vars.debtReserveCache.variableDebtTokenAddress).burn(params.user,vars.actualDebtToLiquidate,vars.debtReserveCache.nextVariableBorrowIndex)`
- In `LiquidationLogic._burnDebtTokens`,
  - `vars.debtReserveCache.nextScaledVariableDebt = IVariableDebtToken(vars.debtReserveCache.variableDebtTokenAddress).burn(params.user,vars.userVariableDebt,vars.debtReserveCache.nextVariableBorrowIndex)`
- In `LiquidationLogic._burnDebtTokens`,
  - `(vars.debtReserveCache.nextTotalStableDebt,vars.debtReserveCache.nextAvgStableBorrowRate) = IStableDebtToken(vars.debtReserveCache.stableDebtTokenAddress).burn(params.user,vars.actualDebtToLiquidate - vars.userVariableDebt)`

State variables written after the call(s)

```
        debtReserve.updateInterestRates(
            vars.debtReserveCache,
            params.liquidationAsset,
            vars.actualDebtToLiquidate,
            0
        );
```

- This function call executes the following assignment(s).
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentLiquidityRate = vars.nextLiquidityRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentStableBorrowRate = vars.nextStableRate.toUint128()`
- In `ReserveLogic.updateInterestRates`,
    - `reserve.currentVariableBorrowRate = vars.nextVariableRate.toUint128()`

External call(s)

```
            _burnDebtTokens(params, vars);
```

- This function call executes the following external call(s).
- In `LiquidationLogic._burnDebtTokens`,
    - `vars.debtReserveCache.nextScaledVariableDebt = IVariableDebtToken(vars.debtReserveCache.variableDebtTokenAddress).burn(params.user,vars.actualDebtToLiquidate,vars.debtReserveCache.nextVariableBorrowIndex)`
- In `LiquidationLogic._burnDebtTokens`,
    - `vars.debtReserveCache.nextScaledVariableDebt = IVariableDebtToken(vars.debtReserveCache.variableDebtTokenAddress).burn(params.user,vars.userVariableDebt,vars.debtReserveCache.nextVariableBorrowIndex)`
- In `LiquidationLogic._burnDebtTokens`,
    - `(vars.debtReserveCache.nextTotalStableDebt,vars.debtReserveCache.nextAvgStableBorrowRate) =`

`IStableDebtToken(vars.debtReserveCache.stableDebtTokenAddress).burn(params.user,vars.actualDebtToLiquidate - vars.userVariableDebt)`

```
_liquidatePTokens(usersConfig, collateralReserve, params, vars);
```

- This function call executes the following external call(s).
- In `LiquidationLogic._liquidatePTokens` ,
    - `IPToken(vars.collateralXToken).transferOnLiquidation(params.user,msg.sender,vars.actualCollateralToLiquidate)`

```
_burnCollateralPTokens(collateralReserve, params, vars);
```

- This function call executes the following external call(s).
- In `LiquidationLogic._burnCollateralPTokens` ,
    - `IPToken(vars.collateralXToken).burn(params.user,msg.sender,vars.actualCollateralToLiquidate,collateralReserveCache.nextLiquidityIndex)`

```
IPToken(vars.collateralXToken).transferOnLiquidation(
    params.user,
    IPToken(vars.collateralXToken).RESERVE_TREASURY_ADDRESS(),
    vars.liquidationProtocolFeeAmount
);
```

State variables written after the call(s)

```
userConfig.setUsingAsCollateral(collateralReserve.id, false);
```

- This function call executes the following assignment(s).
- In `UserConfiguration.setUsingAsCollateral` ,
    - `self.data |= bit`
- In `UserConfiguration.setUsingAsCollateral` ,
    - `self.data &= ~ bit`

LiquidationLogic.sol

External call(s)

```
    SupplyLogic.executeSupply(
        reservesData,
        userConfig,
        DataTypes.ExecuteSupplyParams({
            asset: params.liquidationAsset,
            amount: debtCanBeCovered - vars.actualDebtToLiquidate,
            onBehalfOf: params.user,
            referralCode: 0
        })
    );
```

State variables written after the call(s)

```
    userConfig.setUsingAsCollateral(
        liquidationAssetReserveId,
        true
    );
```

- This function call executes the following assignment(s).
- In `UserConfiguration.setUsingAsCollateral` ,
    - `self.data |= bit`
- In `UserConfiguration.setUsingAsCollateral` ,
    - `self.data &= ~ bit`

MintableIncentivizedERC721.sol

External call(s)

```
    MintableIncentivizedERC721._transfer(from, to, tokenId);
```

- This function call executes the following external call(s).
- In `MintableIncentivizedERC721._transfer` ,
    - `rewardControllerLocal.handleAction(from,oldTotalSupply,oldSenderBalance)`

- In `MintableIncentivizedERC721._transfer`,
    - `rewardControllerLocal.handleAction(to,oldTotalSupply,oldRecipientBalance)`

State variables written after the call(s)

```
_userState[from].collaterizedBalance -= 1;
```

```
_userState[to].collaterizedBalance += 1;
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts.

Alleviation

The team updated the code in commits `87b1ea10496ad4947cb65d4a515313c8b7aa7474` and `7cc940ce18b0a45774948f6d4e86735754d23343`.

SLB-01 | REDUNDANT `else` CLAUSE

Category	Severity	Location	Status
Logical Issue	Minor	contracts/protocol/libraries/logic/SupplyLogic.sol (base): 467~470	Resolved

Description

Refer to the `SupplyLogic.executeUseReserveAsCollateral()` method is used to set whether the ERC20 asset can be collateralized, and there is no similar statement for the ERC721 asset, we think the ERC721 asset the "else" statement is redundant.

Recommendation

We recommend removing the redundant `else` clause.

Alleviation

The team heeded our advice and resolved this issue in commit `aec6ed0ddda43ad3cfbd359c9ffd0d82f45ed6d7` .

TOK-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/tokenization/NToken.sol: 75, 76; contracts/protocol/tokenization/PToken.sol: 75, 76	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
75        (bool success, ) = newImplementation.delegatecall(data);
```

- `newImplementation` is not zero-checked before being used.

```
75        _treasury = treasury;
```

- `treasury` is not zero-checked before being used.

```
76        _underlyingAsset = underlyingAsset;
```

- `underlyingAsset` is not zero-checked before being used.

```
75        _treasury = treasury;
```

- `treasury` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

The team heeded our advice and resolved this issue in commit `355402a64a9c857d9c13b46d16bf813a3186fd56`.

VLB-01 | REDUNDANT CODE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/protocol/libraries/logic/ValidationLogic.sol (base): 111~119, 135~140	Resolved

Description

The require statement is duplicated in `validateSupplyBase()` with the related statements in `validateSupplyERC20()` and `validateSupplyERC721()` .

Recommendation

Consider deleting it if it is useless.

Alleviation

The team heeded our advice and resolved this issue in commit `aec6ed0ddda43ad3cfbd359c9ffd0d82f45ed6d7` .

PARASPACE-02 | POTENTIAL RISKS OF POOL ESTABLISHMENT

Category	Severity	Location	Status
Control Flow	● Informational		● Acknowledged

Description

If multiple pools are deployed, there may be a risk of code reentrancy. The hacker can call the `withdrawERC721()` method of one of the Pools and use the hook `_checkOnERC721Received` method of the ERC721 receiver to reenter the methods(such as the method `liquidationERC721()`) of other Pools. Please check if multiple Pools are allowed to be deployed at the same time.

Recommendation

We recommend the client ensuring the logical correctness.

Alleviation

The team acknowledged this issue and they replied with the following:

"If we deploy multiple pools then basically every pool proxy contract holds its own storage I guess."

POO-03 | DISCUSSION ON BORROW WITH CREDIT

Category	Severity	Location	Status
Control Flow	● Informational	contracts/protocol/pool/Pool.sol: 340, 375, 425, 462	● Resolved

Description

```
424  //  Pool.sol
425  function acceptBidWithCredit(
426    bytes32 marketplaceId,
427    bytes calldata payload,
428    DataTypes.Credit calldata credit,
429    address onBehalfOf,
430    uint16 referralCode
431  ) external virtual override nonReentrant {
```

Currently, the functions `buyWithCredit()` , `batchBuyWithCredit()` , `acceptBidWithCredit()` , and `batchAcceptBidWithCredit()` can be called by anyone and there is no authorization between caller and `onBehalfOf` .

In our opinion, these functions would be invoked in relevant gateway contracts.

- The functions `buyWithCredit()` and `batchBuyWithCredit()` could be invoked in `WETHGateway.sol` .
- The functions `acceptBidWithCredit()` and `batchAcceptBidWithCredit()` could be invoked in `WPunkGateway.sol` .

```
//  WPunkGateway.sol
function acceptBidWithCredit(
  bytes32 marketplaceId,
  bytes calldata payload,
  DataTypes.Credit calldata credit,
  uint256[] calldata punkIndexes,
  uint16 referralCode
) external nonReentrant {
  ......
  Pool.acceptBidWithCredit(
    marketplaceId,
    payload,
    credit,
    msg.sender,
    referralCode
  );
}
```

Recommendation

We recommend adding caller validation to the four methods.

Alleviation

`[ParaSpace]` : We use `msg.sender` 's funds but all validation is done on `onBehalfOf` , which means that `msg.sender` can pay for this purchase for the others, we think it's fine.

PRO-05 | INCORRECT COMMENTS

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/protocol/libraries/logic/BorrowLogic.sol: 56~57; contracts/protocol/tokenization/NToken.sol: 23~27	● Acknowledged

Description

The title comment for `NToken.sol` is for `PToken`, not `NToken`.

The comments for `executeBorrow()` and `executeRepay()` isolated positions are mentioned, however, there are no isolated positions in the code.

Recommendation

We recommend changing the title comment to reflect `NToken`.

We recommend removing references to isolated positions.

Alleviation

ParaSpace team acknowledged this finding.

OPTIMIZATIONS | PARASPACE - NFT MONEY MARKET

ID	Title	Category	Severity	Status
PRO-04	Unused State Variable	Gas Optimization	Optimization	Acknowledged

PRO-04 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/protocol/libraries/configuration/ReserveConfiguration.sol: 22, 28, 29, 40, 48, 49, 50, 60, 61, 62; contracts/protocol/libraries/logic/LiquidationLogic.sol: 82; contracts/protocol/libraries/paraspace-upgradeability/VersionedInitializable.sol: 78; contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol: 95	● Acknowledged

Description

One or more state variables are never used in the codebase.

Variable `BORROWABLE_IN_ISOLATION_MASK` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
22      uint256 internal constant BORROWABLE_IN_ISOLATION_MASK =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFDFFFFFFFFFFFFFFF; // prettier-
ignore
```

```
12  library ReserveConfiguration {
```

Variable `UNBACKED_MINT_CAP_MASK` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
28      uint256 internal constant UNBACKED_MINT_CAP_MASK =
0xFFFFFFFFFFF000000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // prettier-
ignore
```

```
12  library ReserveConfiguration {
```

Variable `DEBT_CEILING_MASK` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
29      uint256 internal constant DEBT_CEILING_MASK =
0xF0000000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // prettier-
ignore
```

```
12  library ReserveConfiguration {
```

Variable `BORROWABLE_IN_ISOLATION_START_BIT_POSITION` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant BORROWABLE_IN_ISOLATION_START_BIT_POSITION = 61;
```

```
library ReserveConfiguration {
```

Variable `IS_DYNAMIC_CONFIGS_START_BIT_POSITION` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant IS_DYNAMIC_CONFIGS_START_BIT_POSITION = 168;
```

```
library ReserveConfiguration {
```

Variable `UNBACKED_MINT_CAP_START_BIT_POSITION` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant UNBACKED_MINT_CAP_START_BIT_POSITION = 176;
```

```
library ReserveConfiguration {
```

Variable `DEBT_CEILING_START_BIT_POSITION` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant DEBT_CEILING_START_BIT_POSITION = 212;
```

```
library ReserveConfiguration {
```

Variable `MAX_VALID_EMODE_CATEGORY` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant MAX_VALID_EMODE_CATEGORY = 255;
```

```
library ReserveConfiguration {
```

Variable `MAX_VALID_UNBACKED_MINT_CAP` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant MAX_VALID_UNBACKED_MINT_CAP = 68719476735;
```

```
library ReserveConfiguration {
```

Variable `MAX_VALID_DEBT_CEILING` in `ReserveConfiguration` is never used in `ReserveConfiguration`.

```
    uint256 internal constant MAX_VALID_DEBT_CEILING = 1099511627775;
```

```
library ReserveConfiguration {
```

Variable `BASE_CURRENCY_DECIMALS` in `LiquidationLogic` is never used in `LiquidationLogic`.

```
    uint256 private constant BASE_CURRENCY_DECIMALS = 18;
```

```
library LiquidationLogic {
```

Variable `______gap` in `VersionedInitializable` is never used in `ATokenDebtToken`.

```
    uint256[50] private ______gap;
```

```
contract ATokenDebtToken is RebasingDebtToken {
```

Variable `_allowances` in `MintableIncentivizedERC721` is never used in `NTokenMoonBirds`.

```
    mapping(address => mapping(address => uint256)) private _allowances;
```

```
contract NTokenMoonBirds is NToken, IMoonBirdBase {
```

Recommendation

We advise removing the unused variables.

Alleviation

ParaSpace team acknowledged this finding.

FORMAL VERIFICATION | PARASPACE - NFT MONEY MARKET

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

Verification of ERC-20 compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-balanceof-succeed-always	Function `balanceOf` Always Succeeds
erc20-balanceof-correct-value	Function `balanceOf` Returns the Correct Value
erc20-balanceof-change-state	Function `balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	Function `allowance` Always Succeeds
erc20-allowance-correct-value	Function `allowance` Returns Correct Value
erc20-allowance-change-state	Function `allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	Function `approve` Prevents Giving Approvals For the Zero Address
erc20-approve-succeed-normal	Function `approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	Function `approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	Function `approve` Has No Unexpected State Changes
erc20-approve-false	If Function `approve` Returns `false`, the Contract's State Has Not Been Changed

Property Name	Title
erc20-approve-never-return-false	Function `approve` Never Returns `false`
erc20-transferfrom-correct-amount	Function `transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	Function `transferFrom` Performs Self Transfers Correctly
erc20-transfer-revert-zero	Function `transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	Function `transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	Function `transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	Function `transfer` Has No Unexpected State Changes
erc20-transfer-exceed-balance	Function `transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	Function `transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If Function `transfer` Returns `false` , the Contract State Has Not Been Changed
erc20-transfer-never-return-false	Function `transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	Function `transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	Function `transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-allowance	Function `transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-balance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-recipient-overflow	Function `transferFrom` Prevents Overflows in the Recipient's Balance

Property Name	Title
erc20-transferfrom-false	If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed
erc20-transferfrom-never-return-false	Function `transferFrom` Never Returns `false`
erc20-totalsupply-succeed-always	Function `totalSupply` Always Succeeds
erc20-totalsupply-correct-value	Function `totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	Function `totalSupply` Does Not Change the Contract's State
erc20-transferfrom-change-state	Function `transferFrom` Has No Unexpected State Changes

Verification Results

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
    - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
    - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.
- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
    - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
    - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Contract MintableERC20 (Source File contracts/mocks/tokens/MintableERC20.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-correct-amount	Inconclusive	
erc20-transferfrom-correct-amount-self	Inconclusive	
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-change-state	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract MockAToken (Source File contracts/mocks/tokens/MockAToken.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Contract stETH (Source File contracts/mocks/tokens/stETH.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract MintableDelegationERC20 (Source File contracts/mocks/tokens/MintableDelegationERC20.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-change-state	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-correct-amount-self	Inconclusive	
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-correct-amount	Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	● True	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Contract ERC20 (Source File contracts/dependencies/openzeppelin/contracts/ERC20.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

APPENDIX | PARASPACE - NFT MONEY MARKET

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | **Securing** the **Web3** World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

