Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Parallel

Para-Space NFT Money Market

Veridise Inc.

January 5, 2023

► **Prepared For:**

Yubo Ruan | Parallel Foundation
parallel.fi

► **Prepared By:**

Jon Stephens
Xiangan He

► **Contact Us:** contact@veridise.com

► **Version History:**

Jan 17, 2023 V1

Contents

1 Executive Summary

From Dec. 12 to Dec. 24, Parallel engaged Veridise to review the security of their Para-Space NFT Money Market. The review covered the on-chain contracts that implement the protocol logic. Veridise conducted the assessment over 4 person-weeks, with 2 engineers reviewing code over 2 weeks on commit `8026a8a`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Summary of issues detected. The audit uncovered 10 issues, 1 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically V-PMM-VUL-001 allows a user to steal the cryptopunks of any user that has specified that the Punk Gateway may buy the punk. In addition, the auditors identified one modererate-severity issue that corresponds to missing validation of the liquidation asset, allowing a user to spend tokens locked in the pool contract.

Code assessment. The Para-Space NFT Money Market is a fork of the AAVE V3 protocol and shares much of the same infrastructure from that project. Like AAVE, the protocol is a pool-based lending protocol that enables lenders to provide liquidity to pools and borrowers to borrow funds from the pools by using collateral. The key difference between the Para-Space NFT Money Market and AAVE is that Para-Space allows users to borrow against NFTs put up as collateral. This extension required new reasoning about the supplying, borrowing, and liquidation logic as well as the introduction of a new coin, the NToken, which users receive upon depositing NFTs into a reserve. Similar to ATokens in the AAVE protocol, NTokens are minted upon deposit of an NFT and they can be used as collateral until they are burned/redeemed/liquidated.

Parallel provided the source code for the Para-Space NFT Money Market contracts for review. A hardhat-based test-suite accompanied the source-code with tests written by the developers. These tests achieve relatively high coverage of the protocol. In addition, the client provided documentation describing the intended behavior for the contracts.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Para-Space NFT Money Market	`8026a8a`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Dec. 12 - Dec. 24, 2022	Manual & Tools	2	4 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	1	1
Medium-Severity Issues	1	1
Low-Severity Issues	7	1
Warning-Severity Issues	1	0
Informational-Severity Issues	0	0
TOTAL	10	3

Table 2.4: Category Breakdown.

Name	Number
Logic Error	3
Validation	2
Theft	1
Specification Error	1
Maintainability	1
Locked Funds	1
Overwritten State	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the on-chain portion of the Para-Space NFT Money Market defined in the following scope. In our audit, we sought to answer the following questions:

- Are assets properly credited to the correct user?
- Is it possible for a user to access an asset they shouldn't?
- Is the pool and NToken properly guarded by reentrancy guards?
- Is it possible for the health factor to change without being checked?
- Can funds be locked within a contract?
- Can a user ever borrow an amount without the appropriate collateral?
- Is it possible for a user to avoid liquidation?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. This audit reviewed the on-chain behaviors contained in the following files of the Para-Space NFT Money Market. As such, Veridise auditors first reviewed the provided documentation to understand the desired behavior of the protocol as a whole. Then, the auditors inspected the provided tests to better understand the intended behavior of the provided contracts at a more granular level. Finally, auditors began a multi-week manual audit of the code assisted by both static analyzers and automated testing.

In terms of the audit, the following files were in-scope:

- paraspace-core/contracts/misc/ERC721OracleWrapper.sol
- paraspace-core/contracts/misc/ParaSpaceOracle.sol
- paraspace-core/contracts/misc/ProtocolDataProvider.sol
- paraspace-core/contracts/protocol/configuration/ACLManager.sol

- paraspace-core/contracts/protocol/configuration/PoolAddressesProvider.sol
- paraspace-core/contracts/protocol/configuration/PoolAddressesProviderRegistry.sol
- paraspace-core/contracts/protocol/libraries/configuration/ReserveConfiguration.sol
- paraspace-core/contracts/protocol/libraries/configuration/UserConfiguration.sol
- paraspace-core/contracts/protocol/libraries/logic/BorrowLogic.sol
- paraspace-core/contracts/protocol/libraries/logic/FlashClaimLogic.sol
- paraspace-core/contracts/protocol/libraries/logic/GenericLogic.sol
- paraspace-core/contracts/protocol/libraries/logic/LiquidationLogic.sol
- paraspace-core/contracts/protocol/libraries/logic/PoolLogic.sol
- paraspace-core/contracts/protocol/libraries/logic/SupplyLogic.sol
- paraspace-core/contracts/protocol/libraries/logic/ValidationLogic.sol
- paraspace-core/contracts/protocol/pool/PoolCore.sol (was Pool.sol)
- paraspace-core/contracts/protocol/pool/PoolParameters.sol (was Pool.sol)
- paraspace-core/contracts/protocol/pool/PoolStorage.sol
- paraspace-core/contracts/protocol/tokenization/base/MintableIncentivizedERC721.sol
- paraspace-core/contracts/protocol/tokenization/NToken.sol
- paraspace-core/contracts/protocol/ui/WPunkGateway.sol
- paraspace-core/contracts/protocol/ui/WETHGateway.sol

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows:

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows:

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-PMM-VUL-001	CryptoPunks Theft	High	Fixed
V-PMM-VUL-002	Missing Liquidation Asset Validation	Medium	Fixed
V-PMM-VUL-003	Health Factor could change after Validation	Low	Open
V-PMM-VUL-004	Anyone can pay the core	Low	Open
V-PMM-VUL-005	HF Calculation considers Inactive Collateral	Low	Open
V-PMM-VUL-006	Cannot rescue ETH	Low	Open
V-PMM-VUL-007	Token does not implement ERC721 Specification	Low	Intended Behavior
V-PMM-VUL-008	No Decimals Validation in Oracle	Low	Open
V-PMM-VUL-009	Move refund to end of executeLiquidateERC721	Low	Open
V-PMM-VUL-010	Data Overwrite in Proxy	Warning	Open

4.1 Detailed Description of Bugs

4.1.1 V-PMM-VUL-001: CryptoPunks Theft

Severity	High	Commit	8026a8a
Type	Theft	Status	Fixed
Files	WPunkGateway.sol		
Functions	supplyPunk, acceptBidWithCredit, batchAcceptBidWithCredit		

The protocol allows CryptoPunks to be supplied to the pool via a custom interface since these NFTs do not adhere to the ERC721 specification. When supplying the punks, however, the WPunkGateway does not check the current owner of the punk. This a malicious user to supply a punk that they do not own if the owner has given the WPunkGateway the right to buy their punk.

```
    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
        );
    }
```

Snippet 4.1: Function that transfers CryptoPunks without checking owner

Impact Since the owner is not checked, Punks could be stolen while in transit to the WPunkGateway. For example, a malicious user could front-run the supplyPunk function to steal the CryptoPunk from the actual owner.

Recommendation Check who owns the CryptoPunk.

4.1.2 V-PMM-VUL-002: Missing Liquidation Asset Validation

Severity	Medium	**Commit**	8026a8a
Type	Validation	**Status**	Fixed
Files	LiquidationLogic.sol		
Functions	_depositETH		

The protocol allows a liquidated user's collateral to be purchased with ETH. When processing the ETH payment, however, the specified liquidation asset is not checked. If someone specifies a non-ETH liquidation asset and still provides ETH to the transaction, the `_depositETH` function will proceed as if the user is intending to perform the liquidation using ETH while the remainder of the protocol proceeds to perform the liquidation with respect to the specified asset.

```
    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
                );
            }
        }
    }
```

Snippet 4.2: The _depositETH function which does not check the liquidationAsset

Impact Since `_depositETH` changes the payer if any ETH is payed along with the transaction, the user can manipulate the specified payer, allowing them to use any tokens owned by the pool to perform perform the liquidation. By taking advantage between the differences between decimals used by different currencies it is possible that someone can perform a liquidation using almost entirely funds locked in the pool.

Recommendation Validate that the liquidation asset is ETH when `msg.value != 0`. If the liquidation asset is not ETH, validate that `msg.value == 0`

4.1.3 V-PMM-VUL-003: Health Factor could change after Validation

Severity	Low	**Commit**	8026a8a
Type	Logic Error	**Status**	Open
Files	SupplyLogic.sol		
Functions	executeWithdrawERC721		

Users are allowed to withdraw collateral as long as their health factor remains above the healthy threshold. After performing the HF validation on a withdraw, however, the reserve may be disabled as collateral if a previous call indicated that the reserve was empty. Since this disabling a reserve as collateral can impact the health factor validation, we would recommend that this be performed before the health factor validation.

```
        if (isWithdrawCollateral) {
            if (userConfig.isBorrowingAny()) {
                ValidationLogic.validateHFAndLtvERC721(
                    reservesData,
                    reservesList,
                    userConfig,
                    params.asset,
                    params.tokenIds,
                    msg.sender,
                    params.reservesCount,
                    params.oracle
                );
            }

            if (newCollateralizedBalance == 0) {
                userConfig.setUsingAsCollateral(reserve.id, false);
                emit ReserveUsedAsCollateralDisabled(params.asset, msg.sender);
            }
        }
```

Snippet 4.3: Snippet of `executeWithdrawERC721` that checks the health factor on a withdraw

Impact If the `newCollateralizedBalance` variable were to be stale, by calling `setUsingAsCollateral` after the validation step, a user's health factor could decrease. This could possibly allow a user to withdraw their collateral while keeping the borrowed funds.

Recommendation Switch the order of `validateHFAndLtvERC721` and `setUsingAsCollateral`

4.1.4 V-PMM-VUL-004: Anyone can pay the core

Severity	Low	**Commit**	8026a8a
Type	Logic Error	**Status**	Open
Files	PoolCore.sol, ParaProxy.sol		
Functions	receive		

The PoolCore contract declares a `receive` function restricting the ability to send funds to the pool to only the `WETH` contract. This `receive` function will never be invoked though because a custom proxy is used to implement the pool logic. In particular, this proxy also declares a `receive` function that allows anyone to transfer funds to the pool.

```
1   receive() external payable {
2       require(
3           msg.sender ==
4               address(IPoolAddressesProvider(ADDRESSES_PROVIDER).getWETH()),
5           "Receive not allowed"
6       );
7    }
```

Snippet 4.4: The `receive` function declared by the PoolCore implementation

Impact Funds could accidentally be sent to the pool, which could then be locked in the pool.

Recommendation If it is intended to restrict the `receive` functionality, place the logic in the `ParaProxy`

4.1.5 V-PMM-VUL-005: HF Calculation considers Inactive Collateral

Severity	Low	**Commit**	8026a8a
Type	Logic Error	**Status**	Open
Files	GenericLogic.sol		
Functions	calculateUserAccountData		

The protocol uses a computed health factor (HF) to determine if a user is eligible for liquidation. When performing this calculation, all collateral include those in non-active and paused reserves are included in the calculation even though collateral in these reserves cannot be liquidated.

Impact If a user has collateral in a reserve that gets marked as inactive or paused, they can continue make borrows against this collateral without worrying about liquidation as any liquidation attempt would fail. This could leave the protocol with no way to liquidate the user if their loan defaults.

4.1.6 V-PMM-VUL-006: Cannot rescue ETH

Severity	Low	**Commit**	8026a8a
Type	Locked Funds	**Status**	Open
Files	PoolLogic.sol		
Functions	executeRescueTokens		

It is possible for user to accidentally transfer ownership of tokens to the pool. If they do so, an admin can intervene to "rescue" the tokens using an api provided by the Pool. This API, however, currently excludes the ability to rescue ETH that may have accidentally been transferred to the pool.

```
1   function executeRescueTokens(
2       DataTypes.AssetType assetType,
3       address token,
4       address to,
5       uint256 amountOrTokenId
6    ) external {
7       if (assetType == DataTypes.AssetType.ERC20) {
8           IERC20(token).safeTransfer(to, amountOrTokenId);
9       } else if (assetType == DataTypes.AssetType.ERC721) {
10          IERC721(token).safeTransferFrom(address(this), to, amountOrTokenId);
11      }
12  }
```

Snippet 4.5: The token rescue logic that excludes ETH

Impact Currently ETH that is transferred to the pool will either be locked.

Recommendation Allow admins to rescue locked ETH from the pool.

4.1.7 V-PMM-VUL-007: MintableIncentivizedERC721 does not implement ERC721 Specification

Severity	Low	**Commit**	8026a8a
Type	Specification Error	**Status**	Intended Behavior
Files	MintableIncentivizedERC721.sol		
Functions	safeTransferFrom		

The MintableIncentivesedERC721 contract does not implement the ERC721 specification because the `safeTransferFrom` function does not call `onERC721Received`.

Impact This can cause an NToken to be locked in a contract that it cannot be recovered from.

Recommendation Implement the ERC721 specification.

Developer Response Due to the potential for re-entrancy the developers have opted not to call `onERC721Received`.

4.1.8 V-PMM-VUL-008: No Decimals Validation in Oracle

Severity	Low	**Commit**	8026a8a
Type	Validation	**Status**	Open
Files	ParaSpaceOracle.sol		
Functions	addAssetSources		

The ParaSpaceOracle allows admins to add or change the pricing oracles used by the protocol. When setting the pricing oracles, no validation is performed to ensure that several invariants assumed by the protocol are maintained.

```
    function _setAssetsSources(
        address[] memory assets,
        address[] memory sources
    ) internal {
        require(
            assets.length == sources.length,
            Errors.INCONSISTENT_PARAMS_LENGTH
        );
        for (uint256 i = 0; i < assets.length; i++) {
            require(
                assets[i] != BASE_CURRENCY,
                Errors.SET_ORACLE_SOURCE_NOT_ALLOWED
            );
            assetsSources[assets[i]] = sources[i];
            emit AssetSourceUpdated(assets[i], sources[i]);
        }
    }
```

Snippet 4.6: The internal function used to overwrite the pricing oracle

Impact Since no validation is performed, an admin could accidentally break one of the protocol's assumptions, leaving it open to attack. Such assumptions include:

- The source is the pricing oracle for the indicated asset
- The source returns a price with respect to ETH
- The source uses 18 decimals

Recommendation Perform appropriate validation when setting the oracle sources

4.1.9 V-PMM-VUL-009: Move refund to end of executeLiquidateERC721

Severity	Low	Commit	8026a8a
Type	Maintainability	Status	Open
Files	LiquidationLogic.sol		
Functions	_depositETH		

The protocol allows a liquidated user's collateral to be purchased with ETH. When processing the payment the protocol will check to determine if the user paid too much and return the excess using `Address.sendValue`.

```
1   function _depositETH(
2       DataTypes.ExecuteLiquidateParams memory params,
3       ExecuteLiquidateLocalVars memory vars
4   ) internal {
5       if (msg.value == 0) {
6           vars.payer = msg.sender;
7       } else {
8           vars.payer = address(this);
9           IWETH(params.weth).deposit{value: vars.actualLiquidationAmount}();
10          if (msg.value > vars.actualLiquidationAmount) {
11              Address.sendValue(
12                  payable(msg.sender),
13                  msg.value - vars.actualLiquidationAmount
14              );
15          }
16      }
17  }
```

Snippet 4.7: The _depositETH function that calls sendValue

Impact This api uses a low-level call to transfer the funds though, opening the protocol to potential reentrancy attacks in the future.

Recommendation The developers should consider moving the refund logic to the end of the liquidation procedure.

4.1.10 V-PMM-VUL-010: Data Overwrite in Proxy

Severity	Warning	**Commit**	8026a8a
Type	Overwritten State	**Status**	Open
Files	ParaProxy.sol		
Functions	updateImplementation		

The ParaSpace protocol uses a custom proxy that allows the functionality of their pool to be updated in the future. The proxy's implementation allows it to inherit behavior from different reference contracts deployed on the blockchain. If the implementation contracts have different storage layouts though, important values may be overwritten. The developers should therefore be very careful to validate that the storage layout of new behaviors is consistent with the proxy's current storage.