By
Quit

Protocol Audit
Issue date
12/06/2022

Overview

The following is a review of ParaSpace, performed by 0xQuit. ParaSpace aims to provide lending similar to Aave or BendDAO. Users can submit NFTs or ERC20 tokens as collateral to borrow against. One feature that is particularly relevant at this time is the ability to deposit a BAYC/MAYC/BAKC token along with ApeCoin, and use that ApeCoin as collateral while staking.

The purpose of this audit is to identify security vulnerabilities, potential gas savings, general best practice recommendations, as well as offer input from a design perspective to minimize community trust requirements.

The audit is based on commit SHA `6c8cc1e0965e5837f5d7ac68debe4da94db4487e`, which has already addressed several concerns raised by earlier audits.

This audit includes a best effort review of potential vulnerabilities, but it is not a guarantee of security. Findings should be used in conjunction with other audits (internal or external), as well as independent verification.

Lastly, this audit is not an endorsement or dismissal of the product. Lending protocols and other sources of leverage are inherently risky, especially if you are inexperienced. Even if the protocol itself is secure, there are many legitimate ways to lose your deposits through liquidation. Use at your own risk.

Thoughts

ParaSpace is a money market protocol, and is in its nascent stages. It is necessarily complex, with many interoperable pieces all combining to provide many layers of functionality. As with any new protocol, I advise users exercise caution in the early days to avoid exploits that may have gone unnoticed during audits. This is even more true for a codebase as large as that of ParaSpace. Lending protocols necessitate many types of approvals, and even giving up custody of your assets. In that regard, ParaSpace is no different.

ParaSpace implements a ParaProxy (a slightly modified Diamond Proxy), with upgradeable facets that control pools and the logic surrounding them. This upgradeability adds an additional requirement of trust: A person with one of the various administrative roles could in theory change the logic that governs any given pool, and steal any approved or deposited NFTs or tokens, or any deposited eth. One potential pattern for guarding against this would be to add a buffer in between declaring an upgrade, and executing said upgrade. This would give users time to withdraw collateral and remove approvals if they do not agree with the upgraded logic.

Findings

Use Custom Errors To Improve Readability And Improve Gas Usage

Severity: QOL Suggestion/Gas Optimization

ParaSpace uses an error library in order to organize errors. All in all, there are currently 130 different errors that can be emitted. Strings are very expensive in solidity, which makes require(condition, error_string) less than ideal. Because ParaSpace errors are short, the cost is not a huge issue - however, the numbered errors also make it very inconvenient to troubleshoot errors while transacting. Instead, opt to use custom errors which cut down on deploy and revert costs, as well as improved descriptors over the numbered system currently in place.

```
9   library Errors {
10      string public constant CALLER_NOT_POOL_ADMIN = "1"; // 'The caller of the function is not a pool admin'
11      string public constant CALLER_NOT_EMERGENCY_ADMIN = "2"; // 'The caller of the function is not an emergency admin'
12      string public constant CALLER_NOT_POOL_OR_EMERGENCY_ADMIN = "3"; // 'The caller of the function is not a pool or emergency admin'
13      string public constant CALLER_NOT_RISK_OR_POOL_ADMIN = "4"; // 'The caller of the function is not a risk or pool admin'
14      string public constant CALLER_NOT_ASSET_LISTING_OR_POOL_ADMIN = "5"; // 'The caller of the function is not an asset listing or pool admin'
15      string public constant CALLER_NOT_BRIDGE = "6"; // 'The caller of the function is not a bridge'
16      string public constant ADDRESSES_PROVIDER_NOT_REGISTERED = "7"; // 'Pool addresses provider is not registered'
17      string public constant INVALID_ADDRESSES_PROVIDER_ID = "8"; // 'Invalid id for the pool addresses provider'
```

There are also several places where the Errors Library is not used, and longer error strings exist, such as in `ParaProxy.fallback()` or the `ParaVersionedInitializable initializer` modifier.

Use transferFrom Over safeTransferFrom When Possible.

Severity: Gas Optimization

`safeTransferFrom` is used throughout the ParaSpace codebase, such as in `ApeCoinStaking._deposit`.

`safeTransferFrom` is identical to `transferFrom`, except that it checks if the receiver is a contract, and if it is, ensures that it implements `onERC721Received`. In cases where you know that the receiver will either be a contract that supports tokens (whether those are ERC20, 721, 1155, or otherwise), then simply using `transferFrom` will allow you to bypass those checks, and save gas as a result.

Use Pre-Increment Operator And Unchecked Blocks Where Possible

Severity: Gas Optimization

There are several instances of loops or other incrementors where over/underflow is not a realistic concern, such as in `NTokenMoonBirds.toggleNesting`. Using pre-increment operators and unchecked blocks here can save a small amount of gas.

Overuse of Reentrancy Guard
Severity: Gas Optimization

The nonReentrant is used very liberally throughout the ParaSpace codebase. Keep in mind that this modifier does add a few thousand gas units, and use it only where needed. Some examples of instances in which it can be removed:

- `MintableIncentivizedERC721.setIsUsedAsCollateral`
- `MintableIncentivizedERC721.safeTransferFrom`
- `NTokenBAYC.depositApeCoin`
- `NTokenBAYC.depositBAKC`

Reentrancy is necessary when making an external call that could in turn call back into your system before state is updated. In cases where the external calls are predictable (i.e. only the ApeCoin contract), or where no external calls are made, the `nonReentrant` modifier can be removed.

Conclusion

No major security vulnerabilities were found in the ParaSpace contracts as a result of this audit, and there is no expectation of unintended loss of assets. The above suggestions will, if implemented, will save significant gas for users of ParaSpace over time. More importantly, the implementation of custom errors will allow ParaSpace to more accurately communicate errors that occur during execution.

ParaSpace naturally inherits vulnerabilities from 3rd party integrations such as X2Y2, Blur, Seaport, and LooksRareExchange. Because these protocols are upgradeable as well, there is a non-zero chance that an exploit that is not currently present becomes present at a later date through no fault of the ParaSpace contracts.

The reliance on Oracles - especially for NFT price data - is a dangerous one. ParaSpace has built what seems to be a robust NFTFloorOracle, but it is difficult to say with certainty that it cannot be manipulated. The same can be said to a lesser extend for external oracles used by ParaSpace.

While there were no critical vulnerabilities found in the code, there is significant trust placed in the ParaSpace team due to administrative controls and contract upgradeability. While it is probably untenable to create a protocol of this magnitude that is not upgradeable, there are measures that can be taken to reduce the amount of trust required.

The ParaSpace team should consider options that segment or remove trust requirements to the best of their ability, especially given the expected value of assets that will be approved or deposited to the protocol.