yAudit Obelisk Review

Review Resources:

- Protocol documentation.

Auditors:

- ljmanini
- adriro

Table of Contents

Review Summary

Obelisk

Obelisk is a lossless lottery protocol that leverages your NFTs as collateral. Users deposit ETH to participate in the lottery, which is sent to the Dinero protocol to accrue yield using the Pirex ETH LSD. Yield accumulated over the round period is then provided as rewards to users.

The contracts of the Obelisk repository were reviewed over 7 days. The code review was performed by 2 auditors between Feb 5 and Feb 11, 2024. The review was limited to the latest commit at the start of the review, this was commit 814eca50992057ed58053c7a3086b1c9c5701632.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/
├── Initializing.sol
├── Obelisk.sol
├── RNGReceiver.sol
├── YieldHubPirexEth.sol
├── arbitrum
│   └── RNGSender.sol
├── interfaces
│   ├── IAutoPxEth.sol
│   ├── IObelisk.sol
│   ├── IPirexEth.sol
│   ├── IRandomizer.sol
│   ├── IRandomizerCallback.sol
│   └── IYieldHub.sol
├── signals
│   └── SObelisk.sol
└── tokens
    └── Papyrus.sol
```

After the findings were presented to the Obelisk team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Obelisk and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Adequate access control is present in user and admin controlled functionality.
Mathematics	Low	We identified issues with the uniformity of the distribution used to select winners.
Complexity	Low	The protocol is quite complex as it integrates with 3 different protocols, in particular the message bridging to fetch the random number.
Libraries	Good	The protocol uses Solmate and the LayerZero base libraries.
Decentralization	Good	Contracts are non-upgradeable and privileged intervention is minimal.
Code stability	Good	The repository was not under active development during the review.
Documentation	Average	Documentation includes explanations and details about most of the workflows of the protocol. However, contracts lack NatSpec comments.
Monitoring	Good	Most actions emit events to monitor activity off-chain.

Category	Mark	Description
Testing and verification	Average	The codebase includes tests, but coverage could be improved.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

1. Critical - Winners selection isn’t fair for all round participants

`_onReceiveRNG()` is responsible for receiving a random number and assigning up to `MAX_WINNERS` winners for the last finalized round. Upon close inspection, we’ve found how winner selection doesn’t occur in a uniform manner. In fact, some participants are more likely to win than others, based solely on their position within the round’s `participants` array.

Technical Details

Running some simulations with the following script, we've found that using the same `seed` and decreasingly taking its modulus from `availableParticipants` to `availableParticipants - 4` introduces a bias which makes the last players in the array win more often than the others.

Impact

Critical. Core functionality of the protocol is fundamentally flawed and exploitable.

Recommendation

Regenerating the seed before each winner is selected, making the extractions totally independent, smooths the winners' distribution to be uniform. The closest approximation of this onchain is to assign `seed` its own hash, to get a series of pseudo-independent values:

```
...
for (uint256 i = 0; i < MAX_WINNERS; ++i) {
    if (availableParticipants == 0) {
        break;
    }

+   seed = uint256(keccak256(abi.encodePacked(seed)));
    winner = seed % availableParticipants;

    cachedPrevIndex = round.participants[totalParticipants - i];
    cachedWinnerIndex = round.participants[winner];
...
```

Developer Response

Fixed in 5ef15813fb57d3ab6f7697fa435772685190913c.

High Findings

1. High - Losers share is unaccounted when total participants is exactly `MAX_WINNERS`

The loser's reward share is added to the carry when the number of winners is less than the maximum, but not when it is exactly the same quantity.

Technical Details

In `_onReceiveRNG(_)`, the implementation considers the case when there are fewer winners than `MAX_WINNERS` and accounts any leftover as carry for next rounds in `sharesCarriedOver`.

```
        if (winnersCount < MAX_WINNERS && claimableShares > principalShares) {
            // if there are less than 5 winners, to-be-undistributed reward is carried over
            uint256 reward = uint256(claimableShares - principalShares);
            sharesCarriedOver += reward.mulDiv(WINNER_REWARD, BPS) * (MAX_WINNERS - winnersCount)
                + reward.mulDiv(LOSERS_REWARD_TOTAL, BPS);
        }
```

The losers share (`LOSERS_REWARD_TOTAL`) is also added to the carry: if it is short on winners, then it also means there are no losers. However, it misses the case when winners are exactly `MAX_WINNERS`. If the number of participants is exactly `MAX_WINNERS`, then it also means there are no losers.

Impact

High. A portion of the rewards could be left unassigned.

Recommendation

Change the condition to also match the case when `winnersCount == MAX_WINNERS`.

```diff
-   if (winnersCount < MAX_WINNERS && claimableShares > principalShares) {
+   if (winnersCount <= MAX_WINNERS && claimableShares > principalShares) {
        // if there are less than 5 winners, to-be-undistributed reward is carried over
        uint256 reward = uint256(claimableShares - principalShares);
        sharesCarriedOver += reward.mulDiv(WINNER_REWARD, BPS) * (MAX_WINNERS - winnersCount)
            + reward.mulDiv(LOSERS_REWARD_TOTAL, BPS);
    }
```

Developer Response

Fixed in 86e59b533c75173409ecc0d9d0bfebb304e9220a.

Medium Findings

1. Medium - Missing `initializer` modifier in RNGSender.sol

The function used to initialize the RNGSender.sol contract lacks protection against re-initialization, allowing the owner to modify the initial configuration.

Technical Details

The `initialize()` function in RNGSender.sol doesn't have the `initializer` modifier, which protects against contract reinitialization.

Even though this function is access protected, it could be used to swap the randomizer address.

Impact

Medium. A malicious or compromised owner account can change the randomizer address.

Recommendation

Add the `initializer` modifier.

Developer Response

Fixed in bc961e7ebd67d482fa017a931c10540f680b83cd.

2. Medium - `RNGSender` can't withdraw its ETH credit on `Randomizer`

`Randomizer` requires contracts using their RNG service to hold a credit line on the contract, from which the request cost is deducted every time a client uses their service.

Technical Details

`Randomizer` allows anyone to top up an arbitrary address's credit line by invoking `clientDeposit(address)`. Similarly, it allows for the owner of the credit line to withdraw its ETH by invoking `clientWithdrawTo(address, uint256)`, which `RNGSender` isn't ever able to call.

Impact

Medium. ETH deposited into `RNGSender`'s credit line on `Randomizer` can never be withdrawn.

Recommendation

Implement a function with proper access control to invoke `randomizer.clientWithdrawTo(...)`.

Developer Response

Fixed in 5f30b06a37d666aa48e0bd261c331001e6e67652.

3. Medium - Pirex fees could break the integration

Fees in the Pirex protocol may disrupt accounting logic and potentially break assumptions in the integration.

Technical Details

Pirex ETH and its autocompounding vault have 3 different fees that apply to the workflows used in the Obelisk integration.

- PirexETH deposit fee: applied when depositing ETH in PirexETH.
- PirexETH normal redemption fee: non-instant redemption can also carry fees.
- AutoPirexETH withdraw penalty: the autocompounding vault has a penalty when shareholders exit the vault.

Even though the documentation says these are all 0%, at the time of writing, the fee for redemptions is 0.03%.

Deposits into Pirex (donations or entries) execute the following path:

- ETH is exchanged for pxETH. Rate is 1:1, but the deposit fee is applied.
- pxETH is deposited in the autocompounding vault, and AutoPxEth shares are minted based on the current price per share (shares are rounded down).

And for withdrawals:

- AutoPxEth shares are redeemed for pxETH. Exchange is based on current price per share, and a penalty is applied (shares are rounded up when using `withdraw(assets)` and assets are rounded down when using `redeem(shares)`).
- Redemption for pxETH mints upxETH on a 1:1 basis, but a redemption fee is applied.

Assumptions that these flows are 1:1 and only yield can accumulate could break the integration. With the current settings, and ignoring differences of eventual rounding of shares, the only fee applied is a 0.03% taken on the received amount of pxETH, which may be relevant for donations, as these do not get any yield at all.

The penalty in AutoPirexETH is particularly interesting: being an ERC4626 vault, the withdrawal process will calculate the excess of shares needed to withdraw the exact amount of assets to comply with the standard. This means that an eventual penalty here could potentially demand more shares than originally minted. For example, when a round is finalized, its principal is calculated using `previewWithdraw()`, which could open the possibility of principal being greater than claimable, if generated yield doesn't cover for the penalty/fee gap. Something similar happens with donations, a donor deposits certain amount of ETH and is minted a number of shares based on that amount, however, while redeeming, the implementation uses `withdraw()` with the original amount of ETH, which could demand more shares than originally minted to cover for the penalty.

Impact

Medium. Most of the fees are currently 0%, except for a small fee in the redemption flow. However, conditions may change if Pirex eventually decides to raise their fees.

Recommendation

Review the impact of any eventual fee in the protocol. A potential solution could be to attach to entries and donations the original amount of shares minted. If the principal ends up being less than the claimable shares, not only does this mean that there is no reward, but that there is a general loss. In this case, the implementation can default to return the original amount of minted shares associated with deposits.

Developer Response

We’re in the middle of negotiating that the withdrawal fee by Obelisk should be zero.

4. Medium - Losers could earn more than winners

In Obelisk, each winner gets 10% of the rewards, while losers share a 45% of the pot. If there aren’t many participants, this could lead to weird situations in which losers earn more than winners.

Technical Details

A portion of the rewards earned in a round go to losers. Each loser will earn a share of the 45% of the pot based on their staked ETH amount over the total ETH staked by losers:

```
        if (roundReward > 0) {
            if (entry.won) {
                // 10% for each winner
                unstakingShares += roundReward.mulDiv(WINNER_REWARD, BPS);
            } else {
                // 45% is split to all losers
                unstakingShares +=
                    roundReward.mulDiv(LOSERS_REWARD_TOTAL,
BPS).mulDiv(entryEth, refRound.stakedEthLosers);
            }
        }
```

This means that if participation is not great, this could lead to cases in which the losers earn more than winners. For example, if participants are 6, then the single loser will earn 45% while each winner gets 10%. Or if there are just two losers who staked a similar amount, then each will get ~22.5%.

Impact

Medium. Fairness could be damaged when participation is low.

Recommendation

A potential solution could be to also include winners in the 45%, so that it is distributed among all participants, making it impossible for a loser to earn more than a winner.

Developer Response

Probably it only happens when v0 is over, with super low probability. And in this case, since the number of losers is lower than winners so losers are actually winners.

Low Findings

1. Low - Validate stake period comes before end of round

Round total duration should be longer than the staking period.

Technical Details

When a new round is created in `startRound()`, the protocol owner specifies the staking duration (`_stakingPeriodInSeconds`) and the total duration (`_durationInSeconds`) of the round.

Staking period is the span of time in which new deposits (entries or an increase of ETH) are accepted. Total duration determines the end time, at which point the round is finalized and winners are drawn.

The difference between these two determine the period in which deposits accrue yield. Setting a total duration shorter than the staking duration would mean that a round can be finalized before the staking deadline hasn't been reached.

Impact

Low. Requires owner mistake.

Recommendation

Validate `_durationInSeconds` is greater than `_stakingPeriodInSeconds`. Optionally, add a minimum span of time that should be dedicated to yield accumulation.

Developer Response

Fixed in a3ea3d57f021de5d81b1e0995b4e952b911742bd.

2. Low - Ensure treasury address is set before sending funds

Protocol fees are sent to the `treasury` address without first checking if it has been correctly initialized.

Technical Details

The `treasury` address in Obelisk can be initially configured via the contract's constructor and the `updateTreasury()` function.

None of these implementations check that the given argument is not the default value. Funds can be potentially lost if fees are distributed before this address is correctly configured.

Impact

Low. Funds can be lost if misconfigured.

Recommendation

In both the constructor and `updateTreasury()`, check that `_treasury != address(0)`.

Developer Response

Fixed in 774a3261c0a37cc99d43e7cbe1fc46d0545d6ba0.

3. Low - Seed with zero value should not be discarded

The implementation of `_onReceiveRNG()` checks that the received random value is different from zero. Although extremely unlikely, this could be a valid value provided by Randomizer.

Technical Details

Random numbers used to seed the draw of a round are generated by the Randomizer protocol and bridged back to Ethereum. When they are received in `_onReceiveRNG()`, the implementation performs a check to ensure this seed is different from zero.

```
    function _onReceiveRNG(bytes32 _slug, bytes32 _value) internal override {
        uint96 roundId = uint96(uint256(_slug));
        uint256 seed = uint256(_value);

        Round storage round = rounds[roundId];

        if (seed == 0) revert InvalidSeed();
```

Since Randomizer provides random numbers in the 256 bit space, zero should still be considered as a valid value.

Impact

Low. Rounds cannot be drawn in the unlikely scenario that zero is selected as the random seed.

Recommendation

Remove the check.

```diff
-   if (seed == 0) revert InvalidSeed();
```

Developer Response

Fixed in c9d74345fdcb5a3617aaecd176b0461662e85c43.

4. Low - `_joinDraw()` may accumulate ETH dust

`_joinDraw()` distributes the call's ETH value uniformly among the number of NFTs staked (which matches the number of entries) a user specifies in calls to `enterDrawBatch()`.

Technical Details

Given that `_joinDraw()` only enforces that `msg.value >= MIN_ETH_JOIN_DRAW * totalNFT` (here), cases in which the forwarded ETH isn't divisible by `totalNFT`, ETH dust will accumulate within the contract.

Impact

Low. Requires users to send a particular amount of ETH and the amount of lost value is minimal.

Recommendation

Consider also verifying `msg.value % totalNFT == 0`.

Developer Response

We know. But because of UX and complexity of reward calculation, we'll go with this.

5. Low - `Obelisk` stops functioning properly once `allEntries.length >= 2 ** 32`

`_joinDraw()` records new participants in the current round by:

1. Recording a user's `entryId = uint32(allEntries.length)`.
2. Pushing the `entryId` to `round.participants`.
3. Pushing a new `DrawEntry` to `allEntries`.
4. Return to 1, while there are still NFTs to be staked.

A user's `entryId` is used further in a draw's lifecycle to either increment the entry's staked ETH or to redeem the entry after a draw has been finalized.

Technical Details

Once `allEntries` reaches `2 ** 32 ~= 4 * 10 ** 9` elements, the participants' `entryId`s will loop back to `0` and forth, effectively mapping participants in different rounds to the same global entry in `allEntries`.

As a consequence, participants that are reassigned a `entryId` will be able to join a draw, but won't be able to:

1. Increase their participation's stake : this check will fail, as `cachedRoundId` is clearly larger than the first participation's `roundId`.
2. Exit the draw once finalized : this check will fail, as the original participation should have already been withdrawn.

Impact

Low. Although it's hard for us to give a precise estimation, ~4B individual entries should occur over a very long time frame.

Recommendation

Estimate the likelihood and speed at which such an event could take place, in function of the expected amount of participants for each round and the cadence at which new rounds are created. Either plan to handle this service degradation gracefully or increase the type size for `round.participants`.

Developer Response

We know. But even if there are always about 10k participants in each round, it should be about 430k rounds to reach it. And even if each round's period is about a week, it takes 8300 years to reach it.

6. Low - Potential denial of service while redeeming from AutoPxEth

Redeeming tiny amounts from AutoPxEth may cause a revert in the vault's implementation, leading to a potential denial of service in Obelisk.

Technical Details

The underlying ERC4626 implementation that AutoPxEth uses reverts if the calculated amount of assets for a given share redemption is rounded down to zero (see AutoPxEth.sol).

This revert can be bubbled to YieldHub.sol and Obelisk.sol when a share redemption is executed, which can happen when users exit the draw or when protocol fees are paid to the treasury address. The former should be covered by the minimum deposit amount (1

ETH), but the latter can be problematic as it is a small portion over the yield generated in the round.

Impact

Low. It would require zero or tiny amounts of shares to trigger the error.

Recommendation

Preview the redemption operation and skip if it is zero.

```
    function redeem(uint256 shares, address to) external onlyObelisk returns (uint256
assets) {
+       IAutoPxEth _autoPxEth = autoPxEth;
+       if (_autoPxEth.previewRedeem(shares) > 0) {
-           assets = autoPxEth.redeem(shares, address(this), address(this));
+           assets = _autoPxEth.redeem(shares, address(this), address(this));
            pirexEth.initiateRedemption(assets, to, false);
+       }

        emit Redeem(shares, assets, to);

        return assets;
    }
```

Developer Response

Fixed in 55d4b3875776da816a4e9a23d86c0441699a6eaf.

7. Low - Unsafe assumption on ERC-721 `tokenId`s

ERC-721 under its “NFT Identifiers” section, specifies that:

> “Every NFT is identified by a unique uint256 ID inside the ERC-721 smart contract. […] callers SHALL NOT assume that ID numbers have any specific pattern to them, and MUST treat the ID as a “black box”. “

Technical Details

`Obelisk` deviates from the standard, as it always treats ERC-721 IDs as `uint32`s. This may prevent the system from employing an NFT collection that uses custom ids, particularly ids larger than `2 ** 32 - 1`.

Impact

Low. The owner must select a collection with this special case. Even if it were employed, either some users wouldn't be able to enter the specific draw, or none would be able to, which would result in a failed draw.

Recommendation

Align with the standard and employ ERC-721 IDs as `uint256`.

Developer Response

We know. Those types of NFT projects are not our target for v0.

Gas Saving Findings

1. Gas - Fail `Papyrus` purchase before executing mint logic

`Papyrus` purchase logic can be modified to revert before any state is written to.

Technical Details

In cases in which `Papyrus.treasury() == address(0)` holds, `buy()` reverts : L46. Notice that this occurs after executing `_mint()`.

Impact

Gas savings.

Recommendation

Execute this check before calling `_mint()`.

Developer Response

Seems to be unnecessary.

2. Gas - Cache storage variable locally to prevent multiple reads from storage

Cache variables read from storage to prevent multiple SLOAD operations.

Technical Details

- `entry.nftId` in `_exitDraw()`.
- `protocolFeeEnabled` in `finalizeRound()`.
- `yieldHub` in `finalizeRound()`.

- `randomizer` in `request()`.
- `treasury` in `buy()`.

Impact

Gas savings.

Recommendation

Cache state in local variables instead of reading again from storage.

Developer Response

Seems to be unnecessary.

3. Gas - Use unchecked math if no overflow risk

There are math operations that can be done unchecked arithmetic for gas savings.

Technical Details

In `redeemDonation()`, `totalDonatedShares` in line 128 can be subtracted using unchecked math since it is checked to be greater in line 127.

In `_exitDraw()`, the `roundReward` calculation in line 252 can use unchecked math due to the ternary check that ensures `claimableShares > principalShares`.

In `finalizeRound()`, `reward` in line 305 can use unchecked math since `claimableShares > principalShares` due to the `if` in line 301.

In `_updateDonatedShares()`, `rewardShares` in line 327 can also use unchecked math due to the `if` in line 326.

In `_onReceiveRNG()`, the `reward` calculation in line 380 can be done using unchecked math since `claimableShares` is greater than `principalShares` due to the `if` in line 378.

Impact

Gas savings.

Recommendation

Use unchecked block if there is no overflow or underflow risk for gas savings.

Developer Response

Seems to be unnecessary.

4. Gas - Unnecessary counter in `_exitDraw()`

The implementation of `_exitDraw()` tracks the number of entries using `unstakingNFT`, which is equivalent to the length of the entry ids array.

Technical Details

The `_exitDraw()` function tracks the number of entries being collected using the `unstakingNFT` variable, in order to determine how many Papyrus tokens should be refunded. However, this number is equivalent to just taking the length of the `_ids` array.

Impact

Gas savings.

Recommendation

Remove `unstakingNFT` from `_exitDraw()`.

Developer Response

Seems to be unnecessary.

Informational Findings

1. Informational - `Initializing.sol` has no license identifier

`Initializing.sol` provides no license identifier, which is inconsistent with the rest of the codebase.

Technical Details

Initializing.sol

Impact

Informational.

Recommendation

Specify an adequate license identifier for the `Initialing.sol` contract.

Developer Response

Fixed in 975fbbdf181a9923fd565f52c5c3789b485ceed4.

2. Informational - Incorrect EIP-165 support for ERC-1155 receiver

The implementation of EIP-165 should also signal support for the ERC-1155 Token Receiver.

Technical Details

The implementation of `supportsInterface()` fails to signify support for the `ERC1155TokenReceiver` interface. See “ERC1155TokenReceiver ERC-165 rules” in EIP-1155.

Impact

Informational.

Recommendation

The `supportsInterface()` function should also return true for ERC-165 and ERC1155TokenReceiver interfaces.

```
    function supportsInterface(bytes4 interfaceId) public pure override returns
(bool) {
-       return interfaceId == type(IObelisk).interfaceId;
+       return interfaceId == type(IObelisk).interfaceId ||
+           interfaceId == type(IERC165).interfaceId ||
+           interfaceId == type(IERC1155Receiver).interfaceId;
    }
```

Developer Response

Fixed in 431a588563d9e9dbda52f2da940599b3a2789d22.

3. Informational - Missing events for on-chain actions

There are some instances of functions that fail to emit an event in order to allow off-chain tracking.

Technical Details

- `enableProtocolFee()`: when protocol fees are turned on.
- `_onReceiveRNG()`: once the random number has been bridged back and winners are selected.

Impact

Informational.

Recommendation

Log an event to allow off-chain monitoring.

Developer Response

Seems to be unnecessary.

Final remarks

The Obelisk protocol is a lottery system that uses the prize savings model where user funds are joined together under a common deposit in a yield protocol with the intention of distributing earnings as prizes. To do so, Obelisk integrates with 3 other protocols: Dinero, to earn yield through the Pirex ETH LSD, Randomizer, to provide on-chain random numbers, and LayerZero, for cross-chain messaging.

The provided documentation was clear, exhibiting multiple flow diagrams that cover how Obelisk integrates with other protocols at different stages of its lifecycle. However, complexity of the architecture and its integrations comes with risks. In M-3 we raise our concerns about the delicate state of assumptions over the Pirex ETH fee system. In M-2, a failure to implement withdrawals from the Randomizer credit line could potentially lock protocol funds. Furthermore, in M-4 we identified a flaw in the lottery's design, by which losing the draw is more profitable than winning it.

Descentralization in the Obelisk protocol is greatly valued, as the protocol operates with minimal privileged intervention. Once the admin starts the round, the workflow is mostly permissionless, except for retrying certain operations in the Arbitrum side.

An innocent looking simplification to select winners led to the discovery of a non-uniform distribution used to allocate prizes. Highlighted in C-1, we show how reusing the same seed while shrinking the array of participants created a bias towards certain indices, causing an unfair distribution of rewards.

The Obelisk team confirmed the reported issues and stated that they are working in a new version of the protocol that will include the fixes.