yAudit Obelisk Review

Review Resources:

- code repositories

Auditors:

- Panda
- HHK

Table of Contents

Review Summary

Obelisk

Obelisk provides a protocol to boost NFT collections with ETH staking rewards.

Inspired by Heroglyphs, where you can execute code based on a signature on your block's graffiti, Obelisk uses the same logic for NFTs. It creates a wrapper for existing collections and then allows users to deposit into reward pools. To allow a collection, 100 ETH will be required. The ETH is deposited into a Liquidity Pool and is forever locked. All yields are sent to the Megapools Program.

The contracts of the Obelisk Repo were reviewed over ten days. Two auditors performed the code review between 7 October and 18 October 2024. The repository was under active development during the review, but the review was limited to the latest 74ca9c65a1d93d2b0fbd0c15225ea500a0291353 for the Obelisk repo.

Scope

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

```
src
├── services
│   ├── HCT.sol
│   ├── InterestManager.sol
│   ├── StreamingPool.sol
│   ├── liquidity
│   │   ├── BaseDripVault.sol
│   │   ├── ChainMoney.sol
│   │   └── apxETHVault.sol
│   ├── nft
│   │   ├── NFTPass.sol
│   │   ├── ObeliskHashmask.sol
│   │   ├── ObeliskNFT.sol
│   │   ├── ObeliskRegistry.sol
│   │   └── WrappedNFTHero.sol
│   └── tickers
│       ├── GenesisTokenPool.sol
│       ├── LiteTicker.sol
│       ├── Megapool.sol
│       └── WrappedGnosisToken.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	Average	While most access controls are implemented properly, there are some issues, like any user being able to forfeit another user's rewards in ObeliskNFT.
Mathematics	Good	Most mathematical operations appear correct, and issues have been found in the calculation of the shares.
Complexity	Good	The codebase is generally well-structured and not overly complex.
Libraries	Average	Some external libraries are used appropriately, but there are opportunities for better integration, like using Cowswap for swaps.
Decentralization	Good	The protocol appears to be designed with decentralization in mind, with appropriate use of governance and distributed systems.
Code stability	Low	Several critical and high severity issues were found, indicating the code is not yet stable for production use.
Documentation	Low	The code has very little to no documentation.
Monitoring	Average	Some events are emitted for key actions, but there's room for improvement in comprehensive logging and monitoring.
Testing and verification	Low	The audit noted a lack of comprehensive testing, particularly for edge cases and potential attack vectors.

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 - Any user can forfeit another user's rewards

The `ObeliskNft` contract allows any user to claim with the flag ignore rewards, resulting in lost rewards.

Technical Details

The function `claim()` of the `ObeliskNft` allows claiming the rewards available from the different pools an NFT id was staked in.

Inside the function, there is a `_claimRequirements()` check that will return `true` or `false` depending on whether the user is the owner of the NFT for the `WrappedNFTHero` implementation and whether the user is the owner and receiver for the `ObeliskHashmask` implementation. Then, the function will loop through the pools and call `claim()` on them, with the tag `_ignoreRewards` set to the opposite of the previous result. This means that if the user calling is not the owner, then the tag will be set to `true`.

The issue is that on the two pool implementations, `MegaPool.sol` and `GenesisTokenPool.sol`, when the tag is set to true, the rewards will be forfeited and distributed back to the contract.

This means any user can force others to forfeit their rewards with just a simple function call.

POC:

There is already a test in the code that shows the claim with the tag set to `true`.

Then there is a test that shows how rewards are distributed back to the contract instead of the user when tag is set to `true` for `GenesisTokenPool.sol`.

Impact

Critical.

Recommendation

Consider reverting if `_claimRequirements()` returns `false` or always set the `_ignoreRewards` tag to `false`.

Developer Response

Resolved: 458ce2d46c53147748158aa7c1d35d5971df7e7b

2. Critical - `_claim` function should update `userYieldSnapshot`

The `userYieldSnapshot` mapping tracks the accumulated yield per token for each user at the time of their last interaction (deposit, withdrawal, or claim). On a claim, the value should be updated to reflect the distribution of rewards, but it's not.

Technical Details

While the `_afterVirtualDeposit` and `_afterVirtualWithdraw` are updating `userYieldSnapshot`, the `claim` function doesn't. This introduces the opportunity to claim multiple times. With the `userYieldSnapshot` not updated, the `sendingReward` remains positive and continually grows.

```
File: Megapool.sol
115:
116:   function _claim(address _holder, bool _ignoreRewards) internal nonReentrant {
117:     INTEREST_MANAGER.claim();
118:     uint256 currentYieldBalance = REWARD_TOKEN.balanceOf(address(this));
119:
120:     if (totalShares > 0) {
121:       yieldPerTokenInRay = yieldPerTokenInRay + ShareableMath.rdiv(_getNewYield(),
totalShares);
122:     } else if (currentYieldBalance != 0) {
123:       REWARD_TOKEN.transfer(owner(), currentYieldBalance);
124:     }
125:
126:     uint256 last = userYieldSnapshot[_holder];
127:     uint256 curr = ShareableMath.rmul(userShares[_holder], yieldPerTokenInRay);
128:
129:     if (curr > last && !_ignoreRewards) {
130:       uint256 sendingReward = curr - last;
131:       REWARD_TOKEN.transfer(_holder, sendingReward);
132:     }
133:
134:     yieldBalance = REWARD_TOKEN.balanceOf(address(this));
135:   }
```

Impact

Critical.

Recommendation

Update `userYieldSnapshot` on a claim.

Developer Response

Resolved: 11ed1315205a589195ce181930922134005bb6f2

High Findings

1. High - `removeFromCollection()` will always revert

The function `removeFromCollection()` will always revert because of a double withdrawal, making it impossible for users to withdraw their deposits.

Technical Details

The function `removeFromCollection()` is supposed to withdraw the deposit from the user if the collection he deposited for didn't reach the `REQUIRED_ETH_TO_ENABLE_COLLECTION`.

When withdrawing, the function first withdraws from `DRIP_VAULT_ETH` and then tries to transfer the ETH amount to the user. However, the current implementation of `DRIP_VAULT_ETH` uses apxETH and directly transfers the token to the user withdrawing. This means the Registry will not receive any ETH and thus will not be able to transfer the ETH to the user, making the transaction revert.

Users will be unable to withdraw their deposit unless they directly send the same amount in ETH before the call to make the ETH transfer succeed.

Impact

High.

Recommendation

Remove the ETH transfer or transfer the axpEth to the registry instead of the user and unwrap the token into ETH.

Developer Response

Resolved: c66d74ccad065760036c8ea57b6712a1a9ca8038

2. High - `InterestManager` will miss some rewards

The `interestManager` might receive apxEth directly from the drip vaults when users deposit and withdraw. Because it doesn't take into account its apxEth balance, then these tokens will not be distributed.

Technical Details

The `InterestManager` stores its rewards only when its internal function `_claimFromServices()` is called. This function will claim rewards from the `streamingPool`, the DAI drip vault, and the ETH drip vault (apxETH).

However, there is a case where the `InterestManager` may receive rewards outside of this internal function call. When a user who has deposited his funds into the ETH drip vault using the Registry withdraws his funds, the drip vault will send the interest generated to the `InterestManager`.

Because the `interestManager` doesn't check its apxETH balance, it will not consider the tokens received from people withdrawing their tokens on the Registry. It will not distribute them to the megaPools.

Impact

High.

Recommendation

Remove the interest transfer from the drip vaults or check the balance of apxETH when determining rewards available in the `InterestManager`.

Developer Response

Resolved: 9d9464f3e5ecade63318e9b7b53bf105d7433901

3. High - Can't virtual deposit into tickers when using `ObeliskHashmask`

The `ObeliskHashmask` is not registered on the registry, which will result in the virtual deposits reverting.

Technical Details

The `ObeliskHashmask` contract is a special version of the Obelisk NFT that allows the Hashmask NFT to be used with Obelisk. It will be deployed on its own and not from the registry.

When depositing into the `LiteTickers` contracts, they check if the NFT calling is registered on the registry. This is done by calling `isWrappedNFT()` . The `isWrappedNFT` mapping is updated when a wrapped NFT is deployed from the registry.

Because the `ObeliskHashmask` is not deployed from the registry, it won't be added to the mapping, and when a user tries to deposit into a ticker, it will revert. They will pay the fee to link and rename for nothing.

Impact

High.

Recommendation

Add an `onlyOwner()` function to add pools to the `isWrappedNFT()` mapping on the registry contract.

Developer Response

Resolved: 5e0abf1718084cebec17a86d5b6ab0a751a1f559

Medium Findings

1. Medium - swap with `amountOutMinimum` set to zero is not recommended

Technical Details

While the risk of a sandwich attack is low due to the high liquidity of the WETH<>DAI pool and the relatively small amounts being swapped, setting `amountOutMinimum` to 0 still exposes the contract to potential losses from price manipulation or extreme market volatility. In the worst-case scenario, the swap could result in receiving significantly less WETH than expected, leading to reduced rewards for users and potentially undermining the integrity of the reward distribution system.

Impact

Medium.

Recommendation

You could explore the possibility of using cowswap orders, but that would require changes in how you distribute rewards. It will protect the protocol from MEV and get a better price than the hardcoded DAI<>ETH pool. An example can be found here.

Developer Response

Resolved: bc2088a93b793c642c277940e751bf04e1ef60dc & a81054408cfff0737e87b4faf430b000b0fc2ba8

2. Medium - A malicious user can `wrap()` then `unwrap()` until there is no more free slots

In the `WrappedNFTHero` contract, some users can wrap their NFT for free. Since it's also free to unwrap, one malicious user could wrap and unwrap until all the free slots have been used. Forcing upcoming users to pay to wrap.

Technical Details

In the function `wrap()`, the user can wrap his NFT for free if his ID is odd or not, depending on the constructor results and if `freeSlots != 0`. Once wrapped, the variable `freeSlots` is reduced by one.

Once wrapped, a user can call the function `unwrap()` for free to get back his original NFT.

A malicious user could spam `wrap()` then `unwrap()` until the `freeSlots == 0` and force users who were supposed to pay for free.

Impact

Medium.

Recommendation

Consider allowing free wrapping only once per NFT ID or add a fee and/or a delay to `unwrap()`.

Developer Response

resolved: 6b76f778102a0d78c94c40b6f601b889225005fd

3. Medium - The HCT multiplier will not update automatically

The HCT rewards are determined using a multiplier that can change over time. However, it is only updated when transferring the token, not automatically, resulting in fewer rewards.

Technical Details

When wrapping for a `WrappedNFTHero` or when receiving one, the function `_update()` will call the `HCT` token and increase the rewards rate to be received by this NFT holder using a `multiplier`.

This `multiplier` is the result of `getWrapperMultiplier()` which will change over time depending if the collection is a premium collection and when it was deployed. Every year it will increase, the older the collection the greater the `multiplier`.

If a user wrapped or received an NFT before the new year passed, his `multiplier` will be less. When the new year comes in, he won't see his `multiplier` increase and thus won't earn more rewards. He must transfer his `WrappedNFTHero` to himself to update his `HCT` rewards rate.

If users are not proactive, they might miss some rewards compared to new users who will directly get the latest `multiplier`.

Impact

Medium.

Recommendation

Consider Proactively calling `getWrapperMultiplier()` on the `HCT` side to determine rewards. This will probably require changes to the contract architecture.

Or consider documenting this effect and showing a user on the frontend when they can update their `multiplier`.

Developer Response

Resolved: b3151a393e79d6322573db0acfcb638cae3df880.

4. Medium - ChainMoney deposits will revert

A potential underflow in the `ChaiMoneyVault` will make deposits revert.

Technical Details

When calling the function `deposit()` it will check if the `INPUT_TOKEN` is different than eth.

The `ChaiMoneyVault` will differ; the amount deposited will be computed by subtracting the `totalDeposit` from the current DAI balance. However, since the DAI deposited are wrapped for chai, the balance will be less every time the deposit is less than the current `totalDeposit`, resulting in an underflow and a revert of the deposit.

Impact

Medium.

Recommendation

Since the registry is trusted, don't compare the deposit against the `totalDeposit`.

Developer Response

Resolved: 1781a11322a9465acc1dc4b99348571fcfd01036

Low Findings

1. Low - `BaseDripVault` `receive()` function can be removed

The function shouldn't be used.

Technical Details

Do not allow plain ETH transfers to the contract. Remove the `receive()` function. ETH would be stuck in the contract if transferred without using the `deposit()` function.

Impact

Low.

Recommendation

Remove the function.

Developer Response

Resolved: e4df808289475e24365c070ef1ff94b0caf775ca

2. Low - Review Math formula in `HCT`

The math formula in HCT seems wrong. Once simplified, it makes no use of the `totalPower`

Technical Details

```
File: HCT.sol
55:     _userInfo.power = uint128(totalPower);
56:     _userInfo.totalMultiplier = uint128(totalMultiplier);
57:     _userInfo.multiplier = totalPower == 0 ? 0 :
uint128(Math.mulDiv(totalMultiplier, PRECISION, totalPower));
```

```
File: HCT.sol
100:     uint256 rateReward = Math.sqrt(_userInfo.power * _userInfo.multiplier) / 1
days;
```

Now, if we replace `_userInfo.multiplier`.

```
uint256 rateReward = Math.sqrt(_userInfo.power * totalMultiplier * PRECISION /
totalPower) / 1 days;
```

`_userInfo.power` being `totalPower` when we simplify we get:

```
 uint256 rateReward = Math.sqrt(totalMultiplier * PRECISION) / 1 days;```
```

Impact

Low. The math formula seems wrong.

Recommendation

Review the math formula.

Developer Response

Resolved: 3e79a2f813f60a82a1f21a59c9896f17a3399511

3. Low - HCT tokens can't be claimed

The HCT tokens accumulate over time so that users can rename an NFT. After discussing this with the team, the protocol will provide a market for buying and selling HCT. However the `HCT` contract lacks a public claim function for users to own the tokens.

Technical Details

The `HCT` contract currently lacks a public `claim` function that would allow users to claim their accumulated HCT tokens manually. The contract implements an internal `_claim` function, which is called automatically when adding or removing power, or when using tokens for renaming. However, users cannot claim their tokens without performing one of these actions.

Impact

Low.

Recommendation

Add a public claim function.

Developer Response

resolved: b6a3bf51e23429342ffb93a34ab2813acc12cf3f

4. Low - MegaPool shares can be removed

Shares will always be equal to the virtual deposit amount. They can be removed.

Technical Details

1 In the `_afterVirtualDeposit` function, the calculation of `addedShare` will always return 1e18:

```
File: Megapool.sol
48:     uint256 addedShare = 1e18;
49:
50:     virtualBalances[_holder] += DEPOSIT_AMOUNT;
51:
52:     if (totalShares > 0) {
53:       addedShare = (totalShares * DEPOSIT_AMOUNT) / totalVirtualBalance;
54:     }
```

This calculation results in: - First deposit: `10**18 = 1.0e18` shares, total shares: 10e18 - Second deposit: `10**18 * 10**18 / 10**18 = 1.0e18` shares, total shares: 2 * 10e18 - Third deposit: `2*10**18 * 10**18 / 2*10**18 = 1.0e18` shares, total shares: 3 * 10e18 - Fourth deposit: `3*10**18 * 10**18 / 3*10**18 = 1.0e18` shares, total shares: 4 * 10e18

1 In `_afterVirtualWithdraw`:

```
File: Megapool.sol
78:     uint256 newShare = 0;
79:     uint256 holderBalance = virtualBalances[_holder];
80:
81:     if (totalShares > 0 && holderBalance > 0) {
82:       newShare = (totalShares * holderBalance) / totalVirtualBalance;
83:     }
84:
```

- Before withdrawal: 4e18 shares, 4e18 tokens deposited.
- Imagine a user with two deposits: 2e18 shares.
- After withdrawal of one deposit, newShares = (4e18 * 1e18) / 4e18 = 1e18.
- Final state: 1e18 shares for the user.

No other part of the codebase changes the `totalVirtualBalance` or the number of `shares`.

Additionally, if changes were made, the share ratio was to fluctuate, the `_afterVirtualWithdraw` will need to be updated as the current implementation will always:

- Reset the user's share value. This means if the user deposited earlier and had a better share value than the current one, withdrawing a part of his deposit will reset his ratio to a new ratio. E.g., Deposited when one share equals one token, now the ratio is one share equals two tokens; if the user deposited 3 NFTs and received three shares and now withdraws 1 NFT, instead of getting his shares reduced by 1/3 (left with two shares), his share ratio is recomputed and leaves him with only one share. When recalculating the shares, the `totalShares` and `totalVirtualBalance` are not reduced, while the `holderBalance` is. This means the user will always receive fewer shares than he should have.

Impact

Low

Recommendation

1. Remove the share-based system entirely.
2. Use the `virtualBalances` mapping directly for reward calculations.
3. Simplify the contract by removing `totalShares`, `userShares`, and related functions.
4. Update the yield distribution logic to use virtual balances instead of shares.

Developer Response

Resolved: db3c258aee78aeb79214bced4a35e5899fff2261

5. Low - Pirex ETH deposits can be paused

The pirex ETH contract used when ETH is added to a collection can be paused.

Technical Details

When the pirex eth contract is paused, the `addToCollection()` function will systematically revert. As a result, users won't be able to contribute ETH to collections.

Impact

Low.

Recommendation

Make sure transactions aren't broadcasted if `PirexEth` is paused.

Developer Response

Acknowledged.

6. Low - Rewards notified before the first deposit will be lost

Technical Details

When `notifyRewardAmount()` is called for the first time it will start a new rewards period. However, all rewards distributed before the first deposit will be lost. Because the reward period that will start on `notifyRewardAmount()` and not on the first deposit.

Impact

Low.

Recommendation

You could reset the `unixPeriodFinish` on the first deposit when `totalSupply == 0 && rewardPerTokenStored == 0 && rewardRate > 0`.

Developer Response

Resolved: 0cc1f855896915ffc12fbccd152c123727fdd956

Gas Saving Findings

1. Gas - Inefficient interest calculation in `ChaiMoneyVault`

In the `ChaiMoneyVault` contract, the `_beforeWithdrawal` and `claim` functions currently calculate the interest gained using a full withdrawal method. This approach is gas-inefficient and can be significantly optimized.

Technical Details

The `_beforeWithdrawal` and `claim` functions in the `ChaiMoneyVault` contract are currently assessing interest gains by performing a full withdrawal of Chai tokens. This method is unnecessarily gas-intensive and can be replaced more efficiently.

A more gas-efficient solution involves reading the `chi` value from the `pot` contract. The relationship between Chai and Dai is defined as:

```
1 Chai = 1 Dai * Pot.chi
```

By utilizing this relationship, we can calculate the current Dai value of Chai tokens without performing any token transfers. This approach allows for accurate interest calculation with significantly reduced gas costs.

Impact

Gas savings.

Recommendation

1. Add the Pot contract as an immutable state variable and read the `chi` value directly from it.
2. Update the `_beforeWithdrawal` and `claim` functions to use the `chi` value for interest calculations without performing full withdrawals.
3. Only withdraw the exact amount of Chai tokens needed for the requested operation.

Use this function to calculate interest in `_beforeWithdrawal` and `claim`, and only perform withdrawals for the required token amount.

The contract will achieve significant gas savings and improved overall efficiency by implementing these changes.

Developer Response

Resolved: 82077bd29e0386c78e1550a956c6bdb3501a0a9e & d72ac03b03c84f3c894fa2990f640d1359fb8054 & d0f492cd9cc03e837f092d9e2625492d768405c8

2. Gas - Give max DAI allowance to Uniswap router

Approving the exact amount for each swap transaction with the Uniswap Router is less efficient than setting a maximum allowance once.

Technical Details

The DAI token contract (0x6b175474e89094c44da98b954eedeac495271d0f) includes an optimization for maximum allowances. When the allowance is set to the maximum uint256 value, the contract doesn't subtract from the allowance, saving gas on subsequent transactions.

```
File: DAI: 0x6b175474e89094c44da98b954eedeac495271d0f
129:         if (src != msg.sender && allowance[src][msg.sender] != uint(-1)) {
```

Approving usage for DAI max value will be more gas efficient.

Impact

Gas savings.

Recommendation

Set the maximum allowance for the Uniswap Router in the constructor.

Developer Response

resolved: 020f9f649481bbeb683b65d212e4b89acaf87720

3. Gas - Inefficient DAI to APX_ETH conversion in `InterestManager`

The `claim` function in the `InterestManager` contract triggers a conversion of DAI to APX_ETH regardless of the amount of DAI claimed. This process involves swapping DAI for WETH, unwrapping WETH to ETH, and depositing ETH into Pirex ETH. This operation is performed without considering a minimum threshold, potentially leading to inefficient use of gas for small amounts.

Technical Details

```
File: InterestManager.sol
144:     DRIP_VAULT_DAI.claim();
145:     uint256 daiBalance = DAI.balanceOf(address(this));
146:     if (daiBalance == 0) return 0;
147:     // @audit: should trigger based on a minimum amount.
148:     TransferHelper.safeApprove(address(DAI), SWAP_ROUTER, daiBalance);
149:
150:     ISwapRouter.ExactInputSingleParams memory params =
ISwapRouter.ExactInputSingleParams({
151:       tokenIn: address(DAI),
152:       tokenOut: address(WETH),
153:       fee: DAI_POOL_FEE,
154:       recipient: address(this),
155:       deadline: block.timestamp,
156:       amountIn: daiBalance,
157:       amountOutMinimum: 0,
158:       sqrtPriceLimitX96: 0
159:     });
```

The function proceeds with the conversion process if the DAI balance is non-zero without considering whether the amount is significant enough to justify the gas costs of the multiple transactions involved.

Impact

Gas savings.

Recommendation

Implement a minimum threshold for the DAI balance before proceeding with the conversion:

Resolved: fee6a7a8aa67a4d4db5a0ff3e4238d2c91daaaa9

Developer Response

4. Gas - Redundant state variable getters

Getters for public state variables are automatically generated with public variables, so there is no need to code them manually, as it adds unnecessary overhead.

Technical Details

```
File: ObeliskRegistry.sol


346: function getCollection(address _collection) external view override returns
(Collection memory) {
347:     return supportedCollections[_collection];
348:   }
```

ObeliskRegistry.sol#L346

Impact

Gas savings.

Recommendation

Remove `getCollection()`

Developer Response

Resolved – supportedCollections is supposed to be internal

b3c2f4ef765b756d53dce9bd74c59ac291a38522

5. Gas - State variables can be packed into fewer storage slots

If variables occupying the same slot are both written using the same function or by the constructor, a separate Gsset (20000 gas) is avoided. Reads of the variables can also be cheaper.

Technical Details

```
File: InterestManager.sol


// @audit: 1 slot could be saved, by using a different order:
\*
 * uint256 PRECISION; // (256 bits)
 * mapping(address => uint128) pendingRewards; // (256 bits)
 * mapping(uint64 => struct IInterestManager.Epoch) epochs; // (256 bits)
 * address gaugeController; // (160 bits)
 * uint64 epochId; // (64 bits)
 * uint32 epochDuration; // (32 bits)
 * contract IStreamingPool streamingPool; // (160 bits)
 * uint24 DAI_POOL_FEE; // (24 bits)
 * address SWAP_ROUTER; // (160 bits)
 * contract IDripVault DRIP_VAULT_ETH; // (160 bits)
 * contract IDripVault DRIP_VAULT_DAI; // (160 bits)
 * contract IERC20 DAI; // (160 bits)
 * contract IWETH WETH; // (160 bits)
 * contract IERC20 APX_ETH; // (160 bits)
 * contract IPirexEth PIREX_ETH; // (160 bits)
 */


23: uint256 public constant PRECISION = 1e18;
24:   uint24 private constant DAI_POOL_FEE = 500;
25:
26:   uint64 public epochId;
27:   uint32 public override epochDuration;
28:   address public gaugeController;
29:   IStreamingPool public streamingPool;
30:
31:   address public immutable SWAP_ROUTER;
32:   IDripVault public immutable DRIP_VAULT_ETH;
33:   IDripVault public immutable DRIP_VAULT_DAI;
34:
35:   IERC20 public immutable DAI;
```

```
36:   IWETH public immutable WETH;
37:   IERC20 public immutable APX_ETH;
38:   IPirexEth public immutable PIREX_ETH;
39:
40:   mapping(address => uint128) internal pendingRewards;
41:   mapping(uint64 => Epoch) public epochs
```

InterestManager.sol#L23

File: ObeliskRegistry.sol

```
// @audit: 1 slot could be saved, by using a different order:
\*
 * uint256 MINIMUM_SENDING_ETH; // (256 bits)
 * uint256 MIN_SUPPORT_AMOUNT; // (256 bits)
 * uint256 COLLECTION_REWARD_PERCENT; // (256 bits)
 * uint256 BPS; // (256 bits)
 * uint256 maxRewardPerCollection; // (256 bits)
 * mapping(address => struct IObeliskRegistry.Collection) supportedCollections; // (256
bits)
 * mapping(address => struct IObeliskRegistry.CollectionRewards)
wrappedCollectionRewards; // (256 bits)
 * mapping(address => bool) isWrappedNFT; // (256 bits)
 * mapping(string => address) tickersLogic; // (256 bits)
 * mapping(address => mapping(address => struct IObeliskRegistry.ContributionInfo))
userSupportedCollections; // (256 bits)
 * mapping(uint32 => struct IObeliskRegistry.Supporter) supporters; // (256 bits)
 * address HCT; // (160 bits)
 * uint32 SUPPORT_LOCK_DURATION; // (32 bits)
 * uint32 supportId; // (32 bits)
 * address NFT_PASS; // (160 bits)
 * contract IERC20 DAI; // (160 bits)
 * contract IDripVault DRIP_VAULT_ETH; // (160 bits)
 * contract IDripVault DRIP_VAULT_DAI; // (160 bits)
 * address treasury; // (160 bits)
 * address dataAsserter; // (160 bits)
 * uint128 REQUIRED_ETH_TO_ENABLE_COLLECTION; // (128 bits)
 */

20: uint256 private constant MINIMUM_SENDING_ETH = 0.005 ether;
21:   uint256 public constant MIN_SUPPORT_AMOUNT = 1e18;
22:   uint32 public constant SUPPORT_LOCK_DURATION = 30 days;
23:   uint256 public constant COLLECTION_REWARD_PERCENT = 4000;
24:   uint256 public constant BPS = 10_000;
```

```
25:   uint128 public constant REQUIRED_ETH_TO_ENABLE_COLLECTION = 100e18;
26:
27:   address public immutable HCT;
28:   address public immutable NFT_PASS;
29:   IERC20 public immutable DAI;
30:   IDripVault public immutable DRIP_VAULT_ETH;
31:   IDripVault public immutable DRIP_VAULT_DAI;
32:
33:   address public treasury;
34:   address public dataAsserter;
35:   uint32 public supportId;
36:   uint256 public maxRewardPerCollection;
37:
38:   mapping(address => Collection) public supportedCollections;
39:   mapping(address wrappedCollection => CollectionRewards) internal
wrappedCollectionRewards;
40:   mapping(address wrappedNFT => bool isValid) public override isWrappedNFT;
41:
42:   mapping(string ticker => address logic) private tickersLogic;
43:   mapping(address user => mapping(address collection => ContributionInfo)) internal
userSupportedCollections;
44:   mapping(uint32 => Supporter) private supporters
```

ObeliskRegistry.sol#L20

Impact

Gas savings.

Recommendation

Reorder state variables.

Developer Response

resolved: 87c38d1f283ce2cd7b81998dfc82552622ad2535

6. Gas - Structs can be packed into fewer storage slots

Each slot saved can avoid an extra Gsset (**20000 gas**) for the first setting of the struct. Subsequent reads, as well as writes, have smaller gas savings.

Technical Details

```
File: IObeliskRegistry.sol


// @audit: 1 slot could be saved, by using a different order:
\*
 * uint256 totalSupply; // (256 bits)
 * uint256 contributionBalance; // (256 bits)
 * address wrappedVersion; // (160 bits)
 * uint32 collectionStartedUnixTime; // (32 bits)
 * bool allowed; // (8 bits)
 * bool premium; // (8 bits)
 */


39: struct Collection {
40:     address wrappedVersion;
41:     uint256 totalSupply;
42:     uint256 contributionBalance;
43:     uint32 collectionStartedUnixTime;
44:     bool allowed;
45:     bool premium;
46:   }
```

IObeliskRegistry.sol#L39

Impact

Gas savings.

Recommendation

Reorder variables.

Developer Response

resolved: bdb6503200f8a47425436185ea30a61990cf996e

7. Gas - Using `storage` instead of `memory` for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a `memory` variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (**2100 gas**) for *each* field of the struct/array. If the fields are read from the new memory variable, they incur an additional `MLOAD` rather than a cheap stack read. Instead of declaring the variable with the `memory` keyword, declaring the variable with the `storage` keyword and caching any fields that need to be re-read in stack variables will be much cheaper, only incurring the Gcoldsload for the fields read.

Technical Details

```
File: NFTPass.sol

160: Metadata memory metadata = metadataPasses[tokenId];
```

NFTPass.sol#L160

Impact

Gas savings.

Recommendation

Use a storage variable.

Developer Response

Resolved: 994a489dceaaab406534d31a19d845ad1a130d12

8. Gas - Remove or replace unused state variables

Unused state variables should be removed or replaced to save gas.

Technical Details

```
File: StreamingPool.sol

21: uint32 public startEpoch;
```

StreamingPool.sol#L21

```
File: ObeliskHashmask.sol

23: string public constant TICKER_SPLIT_HASHMASK = " ";

24: string public constant TICKER_HASHMASK_START_INCIDE = "0";
```

ObeliskHashmask.sol#L23, ObeliskHashmask.sol#L24

```
File: Megapool.sol

22: uint256 private constant WAD = 1e18;
```

Megapool.sol#L22

Impact

Gas savings.

Recommendation

Remove the not used variables.

Developer Response

Resolved: 994a489dceaaab406534d31a19d845ad1a130d12

9. Gas - NFT creation might not need to refund excess eth

In the current implementation, when the ETH amount to be refunded is smaller than the gas price multiplied by 20,000, the contract still attempts to return this small amount to the user. This process consumes more gas than returned value, leading to inefficiency.

Technical Details

```
File: NFTPass.sol
60:     (success,) = msg.sender.call{ value: remainingValue }("");  // @audit: do not
transfer back if cost more than gas cost.
61:     if (!success) revert TransferFailed();
```

NFTPass.sol#L60-L61

Impact

Gas savings.

Recommendation

Return ETH to the user only if it's worth the additional gas.

Developer Response

Resolved: 21cfa822451df95bb9d5ad0c7ea1796bd33d1040

10. Gas - Optimize `updateReceiverAddress()` and `create()` using function polymorphism

The `updateReceiverAddress()` and `create()` functions in the `NFTPass` contract have parameters that can be optional. Using function polymorphism, we can reduce the calldata sent on-chain and consequently reduce gas usage.

Technical Details

For the create() function, the _receiverWallet and _maxCost parameters are optional. The function can be rewritten as:

```
function create(string calldata _name) external {
  create(_name, msg.sender, type(uint256).max);
}

function create(string calldata _name, address _receiverWallet, uint256 _maxCost)
external payable {
  // ... existing implementation ...
}
```

For the `updateReceiverAddress()` function, we can implement the following polymorphism:

```
function updateReceiverAddress(uint256 _nftId, string calldata _name, address _receiver)
external {
_nftId = identityIds[_name];
updateReceiverAddress(_nftId);
}

function updateReceiverAddress(uint256 _nftId, _receiver) public {
  // ... existing implementation ...
}
```

Impact

Gas savings.

Recommendation

Implement the suggested function polymorphism for both `create()` and `updateReceiverAddress()` functions.

Developer Response

Acknowledged - Won't fix. For some reason etherscan does not supporting well polymorphism (potentially other web-based explorer)

11. Gas - Inline `modifiers` that are only used once to save gas

Consider removing the following modifiers and put the logic directly in the function where they are used, as they are used only once.

Technical Details

```
File: services/nft/ObeliskHashmask.sol


39: modifier onlyHashmaskHolder(uint256 _hashmaskId) {
40:     if (hashmask.ownerOf(_hashmaskId) != msg.sender) revert NotHashmaskHolder();
41:     _;
42:   }

44: modifier onlyHashmaskLinker(uint256 _hashmaskId) {
45:     if (identityReceivers[_hashmaskId] != msg.sender) revert NotLinkedToHolder();
46:     _;
47:   }
```

services/nft/ObeliskHashmask.sol#L39, services/nft/ObeliskHashmask.sol#L44

```
File: services/tickers/GenesisTokenPool.sol


39: modifier onlyCanRefillReward() {
40:     if (msg.sender != address(REWARD_TOKEN) && msg.sender != owner()) revert
NotAuthorized();
41:     _;
42:   }
```

services/tickers/GenesisTokenPool.sol#L39

Impact

Gas savings.

Recommendation

Inline modifiers.

Developer Response

Resolved: a41562b8755345e5034675e0b4a82ef0b160fc5a

12. Gas - State variables are accessed, but the value exists in memory

The state variable has been assigned from a memory variable. It is recommended to use the memory variable instead of the state variable. This can save 100 gas per instance.

Technical Details

```
File: services/InterestManager.sol

181: emit GaugeControllerSet(gaugeController);
```

services/InterestManager.sol#L181

```
File: services/tickers/WrappedGnosisToken.sol

160: defaultLzOption =
OptionsBuilder.newOptions().addExecutorLzReceiveOption(lzGasLimit, 0);
```

services/tickers/WrappedGnosisToken.sol#L160

Impact

Gas savings.

Recommendation

Use the existing in-memory value.

Developer Response

Resolved: abf25599d9b65c2cd28dfdd2b076c16194c4a07f

13. Gas - Useless `_minAmountOut` in `send()`

Technical Details

The function `send()` allows sending `genesisToken` from the Ethereum network back to Arbitrum.

The function has a `_minAmountOut` to protect against slippage; however, there is no fee or any functionality that will change the initial asked amount, as the _debit()`function will just burn the amount asked, making this parameter and the check useless.

Impact

Gas.

Recommendation

Remove the parameter and check.

Developer Response

Resolved: 5b3c79b589da3b4652db5f74f2b6e92e637d05fb

14. Gas - Simplify `_queueNewRewards()` check

Technical Details

In the function `_queueNewRewards()` there is a check to decide if it should call `_notifyRewardAmount()` or not depending if the rewards left to be distributed are greater than the queued rewards.

The calculation could be simplified. Instead of calculating a ratio, the formula then compares it. It could be as simple as `if (_rewards > currentAtNow) {}`.

Impact

Gas.

Recommendation

Simplify the check.

Developer Response

Resolved: f9610d79b6ef39cc8af2aa54e9fc05ecf80a9420

15. Gas - Simplify `_deleteShare()`

Technical Details

The function `_deleteShare()` will delete shares from the `userShares` and then calculate the `userYieldSnapshot`.

- Because it will always remove all the shares, instead of `-=`, which will read then subtract, it could just do `= 0`.
- Since the `userShares` will always be 0, then `userYieldSnapshot` will always be 0, so it could replace the `rmulup()` with `= 0`.

Impact

Gas.

Recommendation

Simplify the function.

Developer Response

Resolved with issue: "MegaPool shares can be removed"

16. Gas - Remove `_maxCost` parameter in `create()`

Technical Details

The function `create()` has a parameter `_maxCost` which is supposed to allow the user to specify a limit of fees he wants to pay to create his NFT pass.

However, since the function is payable and the user will use ether to pay, this parameter is useless. The user could just specify his limit using the amount of ether he is going to send when calling the function.

Impact

Gas.

Recommendation

Remove `_maxCost` and the linked checks.

Developer Response

Resolved: 74f427cfb8a7caf5bf22e735117b175fb9b1f3c8

17. Gas - Useless caching in `wrap()`

Technical Details

The function `wrap()` cache the parameter `_inputCollectionNFTId` inside `caughtDepositNFTID`.

However, the two variables will always have the same value; thus, the `caughtDepositNFTID` is unnecessary.

Impact

Gas.

Recommendation

Remove `caughtDepositNFTID`.

Developer Response

Resolved: 169d67b95f92532f3bc9d3732a78e3fc57f56f9d

18. Gas - Useless check in `_credit()`

Technical Details

In the function `_credit()` it checks if the `_to == address(0)` and then sets `_to = owner()`.

However, this case will never happen as the function `_lzReceive()` will always replace `_to` by `pool` if it's equal to 0 before the `_credit()` call.

Impact

Gas.

Recommendation

Remove the first `if` check in `_credit()`.

Developer Response

Resolved: ea4a23242f83cf049389ecd371057343f5914c2c

19. Gas - Useless call to `_queueNewRewards()` when depositing and withdrawing

Technical Details

The functions `_afterVirtualDeposit()` and `_afterVirtualWithdraw` call the internal function `_queueNewRewards()` with 0 as parameter.

If the queued rewards are high enough, this will result in a new rewards period starting; however, in most cases, this function will just use gas for nothing.

Impact

Gas.

Recommendation

Consider setting up a bot that will call it periodically and save gas for your users.

Developer Response

Acknowledged – Won't modify

20. Gas - call `_updateName()` directly in `link()`

Technical Details

In the function `link()` consider calling the internal function `_updateName()` instead of just `_removeOldTickers()` to directly set the new tickers for the user so they don't have to make two different calls.

Impact

Gas.

Recommendation

Call `_updateName()` instead of `_removeOldTickers()`.

Developer Response

Resolved: 9f673a82d8b878203fd3525bfbecc7ad6b8be88d

21. Gas - Useless array in `_addNewTickers()`

Technical Details

In the function `_addNewTickers()` the variable `potentialTickers` is used to loop through all the potential tickers to deposit in.

Instead of creating an array that will never be read and is only used for its length, consider creating a `uint256` variable.

Impact

Gas.

Recommendation

Replace the array with a `uint256` variable.

Developer Response

Resolved: fce5b55637f05347fb4f43f40ae1cfb5bccbc5d8

22. Gas - Simplify check in `claim()`

Technical Details

In the function `claim()` there is a check `if (contributionBalance == 0 || !collection.allowed) revert NothingToClaim();`.

This check could be simplified by checking if the `contributionBalance != 100 eth` since there will be nothing to claim if the collection doesn't reach 100 eth, and it is impossible to add eth to the collection if it is not allowed.

Impact

Gas.

Recommendation

Simplify the check with the suggestion.

Developer Response

Resolved: 315d038ef226de17127ea06a32c218afacd71ce4

Informational Findings

1. Informational - Typos

Typos found on the code.

Technical Details

Technical Details

```
File: interfaces/IInterestManager.sol


// @audit: Initialized should be Initialized
10: event EpochInitialized(uint64 indexed epochId, address[] megapools, uint128[]
weights, uint128 totalWeight);
```

interfaces/IInterestManager.sol#L10

```
File: services/InterestManager.sol


// @audit: Initialized should be Initialized
88: emit EpochInitialized(epochId, _megapools, _weights, totalWeight);
```

services/InterestManager.sol#L88

```
File: services/nft/ObeliskNFT.sol

// @audit: registred should be registered
41: address registeredUserAddress = identityReceivers[_tokenId];

// @audit: registred should be registered
42: _removeOldTickers(registeredUserAddress, _tokenId, false);

// @audit: registred should be registered
56: address registeredUserAddress = identityReceivers[_tokenId];

// @audit: registred should be registered
57: _removeOldTickers(registeredUserAddress, _tokenId, false);

// @audit: registred should be registered
63: function _removeOldTickers(address _registeredUserAddress, uint256 _tokenId, bool
_ignoreRewards)

// @audit: registred should be registered
71: ILiteTicker(activePools[i]).virtualWithdraw(_tokenId, _registeredUserAddress,
_ignoreRewards);

// @audit: registred should be registered
76: function _addNewTickers(address _registeredUserAddress, uint256 _tokenId, string
memory _name) internal virtual {

// @audit: registred should be registered
91: ILiteTicker(poolTarget).virtualDeposit(_tokenId, _registeredUserAddress);
```

services/nft/ObeliskNFT.sol#L41, services/nft/ObeliskNFT.sol#L42, services/nft/ObeliskNFT.sol#L56, services/nft/ObeliskNFT.sol#L57, services/nft/ObeliskNFT.sol#L63, services/nft/ObeliskNFT.sol#L71, services/nft/ObeliskNFT.sol#L76, services/nft/ObeliskNFT.sol#L91

```
File: services/nft/ObeliskRegistry.sol

// @audit: santized should be sanitized
156: uint256 santizedAmount = msg.value != 0 ? msg.value : _amount;

// @audit: santized should be sanitized
158: if (santizedAmount < MIN_SUPPORT_AMOUNT) revert AmountTooLow();

// @audit: santized should be sanitized
164: amount: uint128(santizedAmount),

// @audit: santized should be sanitized
172: DAI.transferFrom(msg.sender, address(DRIP_VAULT_DAI), santizedAmount);

// @audit: santized should be sanitized
173: DRIP_VAULT_DAI.deposit(santizedAmount);

// @audit: santized should be sanitized
176: emit Supported(supportId, msg.sender, santizedAmount);

// @audit: Collaction should be Collection
254: uint128 totalCollactionClaimedRewards = collectionRewards.claimedRewards;

// @audit: Collaction should be Collection
258: collectionRewards.claimedRewards = totalCollactionClaimedRewards + rewardsToClaim;
```

services/nft/ObeliskRegistry.sol#L156, services/nft/ObeliskRegistry.sol#L158, services/nft/ObeliskRegistry.sol#L164, services/nft/ObeliskRegistry.sol#L172, services/nft/ObeliskRegistry.sol#L173, services/nft/ObeliskRegistry.sol#L176, services/nft/ObeliskRegistry.sol#L254, services/nft/ObeliskRegistry.sol#L258

Impact

Informational

Recommendation

Fix typos

Developer Response

Resolved: ca68d94e1fcc9a12fe07607ead0d00139ba3b086

2. Informational - Precompute address to remove `initHCT()`

Technical Details

The `HCT` has a `initHCT()` function. This is because of a circle reference between multiple contracts.

It could be interesting to precompute the address of one of the contracts before deployment so that `obeliskRegistry` can be set to `immutable,` and `isInitialized` removed.

Impact

Informational.

Recommendation

Consider precomputing the address of one contract and setting it in the constructor instead of `initHCT()`.

Developer Response

Resolved 334e312ac4d456147b28626e27806ab70eff6ecd.

3. Informational - Wrong parameter name in `estimateFee()`

Technical Details

The function `estimateFee()` has a parameter named `_tokenId` but it should be named `_amount`.

Impact

Informational.

Recommendation

Rename the parameter.

Developer Response

Resolved: 41e4f29a52c528bcf63a0952a98e38363c24c4f6

4. Informational - Last user calling `addToCollection()` can pick the `FREE_SLOT_FOR_ODD`

Technical Details

The `FREE_SLOT_FOR_ODD` is determined using the `tx.origin`. It will be the address of the last user calling the `addToCollection` function. This might be unfair as this user could call from an address that would give him free NFTs wrapping.

Since it's hard to make it fully random without calling an oracle, it might be more fair for every user only to use the collection's address, so it's deterministic and fair for everyone.

Impact

Informational.

Recommendation

Only use the collection address.

Developer Response

Resolved: 0763c70737472baea5011b23bcdb9d3e5a50e18c

Final remarks

The Obelisk protocol demonstrates innovative concepts in NFT-based yield generation. However, the audit revealed several critical and high-severity issues that must be addressed before the protocol can be deployed. These include vulnerabilities in reward distribution, potential loss of user funds, and inefficiencies in various operations. The development team should prioritize fixing these issues, improving documentation, and implementing more

comprehensive testing. Additionally, enhancing access controls, optimizing gas usage, and considering more efficient swap mechanisms could significantly improve the protocol's security and performance. For this reason, auditors encourage the protocol to go through another review before deploying it to production.