Lens V3 Core Security Review

Pashov Audit Group

Conducted by: Shaka, ubermensch, ast3ros, merlinboii

January 6th 2025 - February 3rd 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **lens-protocol/lens-v3** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Lens V3 Core

Lens is an open social network where every EVM account acts as a Profile, supporting smart wallets and social features. It is built on modular primitives (Feed, Graph, Group, Namespace) that can be extended with custom Actions and Rules, enabling flexible and customizable interactions within the ecosystem.

Lens Core contains the main contracts that make up the base Lens Protocol. These contracts are envisioned as non-opinionated and flexible, allowing for a wide range of use cases.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - faf3765db6727e5e5e5f5cdd03430870a3cfb0cd

fixes review commit hash - ec11f802a3c8168219dc3725c798562726ef6b70

Scope

The following smart contracts were in scope of the audit:

```
- AccessControlled
- Ownable
- RoleBasedAccessControl
- BaseSource
- ExtraStorageBased
- LensERC721
- MetadataBased
- RuleBasedPrimitive
- SourceStampBased
- interfaces/
- AccessControlLib
- CallLib
- EIP712EncodingLib
- KeyValueStorageLib
- RulesLib
- Feed
- FeedCore
- RuleBasedFeed
- Graph
- GraphCore
- RuleBasedGraph
- Group
- GroupCore
- RuleBasedGroup
- LensUsernameTokenURIProvider
- Namespace
- NamespaceCore
- RuleBasedNamespace
- Errors
- Events
- Types
- Beacon
- BeaconProxy
- Initializable
- Lock
- ProxyAdmin
```

7. Executive Summary

Over the course of the security review, Shaka, ubermensch, ast3ros, merlinboii engaged with Avara to review Lens V3 Core. In this period of time a total of **18** issues were uncovered.

Protocol Summary

Protocol Name	Lens V3 Core
Repository	https://github.com/lens-protocol/lens-v3
Date	January 6th 2025 - February 3rd 2025
Protocol Type	Social Network

Findings Count

Severity	**Amount**
High	2
Medium	11
Low	5
Total Findings	**18**

Summary of Findings

ID	Title	Severity	Status
[H-01]	lastFollowIdAssigned is not set in graph migration	High	Resolved
[H-02]	Non-unique entityId causing source stamp overwrite in Graph	High	Resolved
[M-01]	Encode type does not follow EIP-712 standard for nested structs	Medium	Resolved
[M-02]	Data encoding does not follow the EIP-712 standard	Medium	Resolved
[M-03]	LensERC721 does not implement the ERC165 interface	Medium	Resolved
[M-04]	ProxyAdmin does not accept native tokens in call() function	Medium	Resolved
[M-05]	Source stamp validation is not linked to a specific action or msg.sender	Medium	Resolved
[M-06]	_processPostEditingOnFeed() receives wrong parameters on post editing	Medium	Resolved
[M-07]	Incorrect storage slot used in LensERC721	Medium	Resolved
[M-08]	Users can be forced in a group	Medium	Resolved
[M-09]	Potential bypass of edit post rule validation for quote posts	Medium	Resolved
[M-10]	Lack of upgrade-aware patterns	Medium	Resolved
[M-11]	Source is not reset after primitive entity removal	Medium	Resolved
[L-01]	A wrong event is emitted even if the role is already assigned to the account	Low	Resolved

[L-02]	BaseSource does not allow signature cancellation	Low	Resolved
[L-03]	Last account's follow ID is not reusable	Low	Resolved
[L-04]	changePostRules() does not check if the post is a root post	Low	Resolved
[L-05]	Inconsistent return value in _createPost()	Low	Resolved

8. Findings

8.1. High Findings

[H-01] lastFollowIdAssigned is not set in graph migration

Severity

Impact: Medium

Likelihood: High

Description

The `MigrationGraph._followWithoutChecks` function is intended for internal use during migration to bypass standard checks and directly populate follow relationships. However, it contains an error: it sets the `followId` for a given follow relationship but fails to increment the `lastFollowIdAssigned` counter for the `accountToFollow`.

This omission has the following consequence:

When a subsequent `follow` operation targets the same `accountToFollow`, and the internally generated `followId` (which increments `lastFollowIdAssigned`) happens to match a `followId` that was already used during migration via `_followWithoutChecks`, a collision will occur. It can lead to:

- Overwriting Existing Follow Relationships: A new `follow` operation might overwrite a previously migrated follow relationship if they end up using the same `followId`. This results in data loss.
- Incorrect Data Storage: The `followers` mapping, indexed by `accountToFollow` and `followId`, could become corrupted with incorrect `followerAccount` addresses associated with specific `followId`s.
- Potential Unfollow Issues: Data corruption from ID overwriting could lead to users being unable to unfollow correctly or potentially unfollowing the wrong accounts.

```
function _followWithoutChecks(
      addressfollowerAccount,
      addressaccountToFollow,
      uint256followId,
      uint256timestamp
    )
        internal
    {
        require(followerAccount != accountToFollow, Errors.ActionOnSelf());
        require(followId != 0, Errors.InvalidParameter());
        require(followerAccount != address(0), Errors.InvalidParameter());
        require(accountToFollow != address(0), Errors.InvalidParameter());
        require(Core.$storage(
          Core.$storage

        ).follows[followerAccount][accountToFollow].id == 0, Errors.CannotFollowAgain(
        require(Core.$storage().followers[accountToFollow][followId] == address
          (0), Errors.AlreadyExists());
        Core.$storage().follows[followerAccount][accountToFollow] = Follow
          ({id: followId, timestamp: timestamp});
        Core.$storage().followers[accountToFollow][followId] = followerAccount;
        Core.$storage().followersCount[accountToFollow]++;
        Core.$storage().followingCount[followerAccount]++;
    }
```

- `followId` is set to 0.

```
function follow(
        address followerAccount,
        address accountToFollow,
        KeyValue[] calldata customParams,
        RuleProcessingParams[] calldata graphRulesProcessingParams,
        RuleProcessingParams[] calldata followRulesProcessingParams,
        KeyValue[] calldata extraData
    ) external virtual override returns (uint256) {
        ...
        uint256 assignedFollowId = Core._follow
          (followerAccount, accountToFollow, 0, block.timestamp);
        ...
    }
```

- So the followId is set to be equal `lastFollowIdAssigned` + 1.

```
function _follow(
     addressfollowerAccount,
     addressaccountToFollow,
     uint256followId,
     uint256timestamp
   )
        internal
        returns (uint256)
    {
        ...
        if (followId == 0) {
            followId = ++$storage().lastFollowIdAssigned[accountToFollow];
        ...
    }
```

Recommendations

- Increase the `lastFollowIdAssigned` in `_followWithoutChecks` OR
- Allow users to specify the `followId` in follow function.

[H-02] Non-unique `entityId` causing source stamp overwrite in `Graph`

Severity

Impact: Medium

Likelihood: High

Description

The `Graph` contract creates a collision for source stamps due to non-unique `followId` assignment as the `entityId` across different `accountToFollow`. The source stamp associated with the `followId` can be continuously replaced by new follow or unfollow activities.

Using the follow action as an explanation example:

```
// File: contracts/core/primitives/graph/Graph.sol

function follow(
    ...
) external virtual override returns (uint256) {
    require(msg.sender == followerAccount, Errors.InvalidMsgSender());
    // followId is now in customParams - think if we want to implement this now,
    // or later. For now passing 0 always.
@1> uint256 assignedFollowId = Core._follow
  (followerAccount, accountToFollow, 0, block.timestamp);
@2> address source = _processSourceStamp(assignedFollowId, customParams);
    --- SNIPPED ---
}
```

```
// File: contracts/core/primitives/graph/GraphCore.sol

function _follow(
  addressfollowerAccount,
  addressaccountToFollow,
  uint256followId,
  uint256timestamp
)
    internal
    returns (uint256)
{
    require(followerAccount != address(0), Errors.InvalidParameter());
    require(accountToFollow != address(0), Errors.InvalidParameter());
    require(followerAccount != accountToFollow, Errors.ActionOnSelf());
    require($storage(
      $storage

    ).follows[followerAccount][accountToFollow].id == 0, Errors.CannotFollowAgain(
    if (followId == 0) {
@1>     followId = ++$storage().lastFollowIdAssigned[accountToFollow];
    } else {
    --- SNIPPED ---
}
```

Since `_processSourceStamp` directly uses `followId` as the `entityId`, and `followId` is sequentially assigned per `accountToFollow` (`@1>`), the source stamp for a given `followId` can be overwritten over time, leading to incorrect source stamping.

Consider the following scenario: Assume `lastFollowIdAssigned[Bob]: 0`, `lastFollowIdAssigned[Carol]: 0`

1. Alice follows Bob on App A via the App's graph primitive.
    - The source stamp at this step stores `entityId: 1 => source: A`.
2. David calls the App A's graph primitive and follows Carol while passing a different source to be stamped.
    - The previous source stamp of `entityId: 1` is overwritten by this step, resulting in `entityId: 1 => source B`.

Recommendations

Ensure that the `source` stamp is assigned to unique `entityId`. One possible approach is to incorporate the `accountToFollow` when generating the `entityId`.

Moreover, if the `followId` is intended to be used for further references, it should also be considered for uniqueness.

8.2. Medium Findings

[M-01] Encode type does not follow EIP-712 standard for nested structs

Severity

Impact: Low

Likelihood: High

Description

According to the ERC-712 standard for the type encoding of a struct:

> If the struct type references other struct types (and these in turn reference even more struct types), then the set of referenced struct types is collected, sorted by name and appended to the encoding. An example encoding is `Transaction(Person from,Person to,Asset tx)Asset(address token,uint256 amount)Person(address wallet,string name)`.

However, in the `EIP712EncodingLib.sol` library, the structs referenced by the main struct are not appended to the encoding, making it not compliant with the EIP and resulting potentially in issues with integrators.

```
keccak256(
        "RuleChange(
          addressruleAddress,
          bytes32configSalt,
          RuleConfigurationChangeconfigurationChanges,
          RuleSelectorChange[]selectorChanges
        )"
    ),
```

Recommendations

Append the referenced structs to the encoding to be compliant with the EIP-712 standard.

[M-02] Data encoding does not follow the EIP-712 standard

Severity

Impact: Low

Likelihood: High

Description

According to the ERC-712 standard for the data encoding:

> The encoding of a struct instance is `enc(value₁) ‖ enc(value₂) ‖ … ‖ enc(valueₙ)`, i.e. the concatenation of the encoded member values in the order that they appear in the type. Each encoded member value is exactly 32-byte long.

However, the `EIP712EncodingLib.sol` library uses `abi.encodePacked` to encode the data, which provokes that not all members are encoded in 32-byte long chunks.

```
keccak256(
        "RuleChange(
          addressruleAddress,
          bytes32configSalt,
          RuleConfigurationChangeconfigurationChanges,
          RuleSelectorChange[]selectorChanges
        )"
    ),
```

This can cause issues with integrators that use the EIP-712 standard to sign messages.

Recommendations

Use `abi.encode` instead of `abi.encodePacked` for encoding the data.

[M-03] `LensERC721` does not implement the `ERC165` interface

Severity

Impact: Low

Likelihood: High

Description

According to EIP-721:

> Every ERC-721 compliant contract must implement the ERC721 and ERC165 interfaces

However, the `LensERC721` contract does not implement the `ERC165` interface. The `ERC165` interface is used to check if a contract implements a specific interface, so the absence of this interface may lead to compatibility issues with other contracts that rely on this interface to verify that `LensERC721` is an ERC721 contract.

Recommendations

Add the following function to the `LensERC721` contract to implement the `ERC165` interface:

```
function supportsInterface(bytes4 interfaceId) public view virtual returns
      (bool) {
        return
            interfaceId == 0x01ffc9a7 || // interfaceId for ERC165
            interfaceId == 0x80ac58cd || // interfaceId for ERC721
            interfaceId == 0x5b5e139f;   // interfaceId for ERC721Metadata
    }
```

[M-04] `ProxyAdmin` does not accept native tokens in `call()` function

Severity

Impact: Low

Likelihood: High

Description

The `ProxyAdmin.call()` function allows the owner to pass a `value` parameter for the value that will be sent to the target contract. However, the `call()` function does not have the `payable` modifier and the contract does not have a fallback function to receive native tokens. This means that the contract will revert if the `value` parameter is greater than 0.

Recommendations

Add the `payable` modifier to the `call()` function to allow the contract.

[M-05] Source stamp validation is not linked to a specific action or `msg.sender`

Severity

Impact: Medium

Likelihood: Medium

Description

Some functions of the primitive contracts receive an array of custom parameters. When this array contains a source stamp, the source stamp is validated by the source contract (e.g. `App` contract).

The source stamp data includes the source address, a nonce, and a deadline. This means that this data is not linked to a specific action or msg.sender. So the same signature can be used to validate any action from any address. Additionally, it can cause DoS attacks by front-running user's transactions using their source stamp and using the nonce of the signature.

Recommendations

Add to the source stamp data the action hash and the `msg.sender` address.

[M-06] `_processPostEditingOnFeed()` receives wrong parameters on post editing

Severity

Impact: Medium

Likelihood: Medium

Description

In the `Feed.editPost()` function the `rootPostRulesParams` are used instead of `feedRulesParams` for the `_processPostEditingOnFeed()` function call.

```
function editPost(
        uint256 postId,
        EditPostParams calldata postParams,
        KeyValue[] memory customParams,
        RuleProcessingParams[] memory feedRulesParams,
@>      RuleProcessingParams[] memory rootPostRulesParams,
        RuleProcessingParams[] memory quotedPostRulesParams
    ) external virtual override {
(...)
@>      _processPostEditingOnFeed
  (postId, postParams, customParams, rootPostRulesParams);
        uint256 quotedPostId = Core.$storage().posts[postId].quotedPostId;
        if (quotedPostId != 0) {
            uint256 rootOfQuotedPost = Core.$storage
              ().posts[quotedPostId].rootPostId;
            _processPostEditingOnRootPost(
              rootOfQuotedPost,
              postId,
              postParams,
              customParams,
              quotedPostRulesParams
            );
        }
        uint256 rootPostId = Core.$storage().posts[postId].rootPostId;
        if (postId != rootPostId) {
@>          _processPostEditingOnRootPost
  (rootPostId, postId, postParams, customParams, rootPostRulesParams);
```

Once the issue is known, it might be possible to execute the function by adding the feed rules to the `rootPostRulesParams` instead of using `feedRulesParams`. However, there is the possibility that the same contract is used to manage feed and post rules, resulting in the DoS for the function in the best-case scenario, or a loss of funds in the worst-case scenario.

Consider the following scenario:

- There is a `SimplePaymentFeedAndPostRule` contract that implements both `FeedRule` and `PostRule` interfaces and allows adding rules for payment on post editing.
- Bob creates a rule for post edition that requires a payment of 10 USDC.
- Alice replies to Bob's post.
- Alice wants to edit the post, so she gives maximum allowance to the `SimplePaymentFeedAndPostRule` contract and calls the `editPost` function with empty `feedRulesParams` and `rootPostRulesParams` filled with the expected payment data.
- Alice is not aware that there is also a feed rule that requires a payment of 10 USDC, but as the `rootPostRulesParams` are used instead of `feedRulesParams`, both rules are executed and Alice pays 20 USDC instead of the 10 USDC she expected.

Recommendations

```
-         _processPostEditingOnFeed
- (postId, postParams, customParams, rootPostRulesParams);
+         _processPostEditingOnFeed
+ (postId, postParams, customParams, feedRulesParams);
```

[M-07] Incorrect storage slot used in `LensERC721`

Severity

Impact: Medium

Likelihood: Medium

Description

`LensERC721` contract is meant to be using the keccak of `lens.storage.ERC721` as the storage slot for the `ERC721Storage` struct. However, the declared value of the storage slot corresponds to the keccak of `lens.storage.RuleBasedStorage`.

The keccak of `lens.storage.RuleBasedStorage` is also used by the `RuleBasedFeed` contract. Both contracts are part of the core of the protocol and are meant to be used by developers to extend the protocol. If a new primitive

that inherits from `LensERC721` and `RuleBasedFeed` (or another contract that uses the same storage slot) is created, both contracts will be using the same storage slot, which can lead to unexpected behavior.

Recommendations

```
/// @custom:keccak lens.storage.ERC721
-
-    bytes32 constant STORAGE__ERC721 = 0x5d84583cb768017b44ca3aec8199901a24d17ed118ff
+
+    bytes32 constant STORAGE__ERC721 = 0x9773440c5f3d31ef6a1be068fec8ef97f4aa1ba801bb
```

[M-08] Users can be forced in a group

Severity

Impact: Low

Likelihood: High

Description

The `Group` contract allows accounts to be added as members without their consent, either by admins with `PID__ADD_MEMBER` permission or by anyone if the account meets the processAddition rules. While users can leave a group via `leaveGroup`, there is no mechanism to prevent being re-added repeatedly against their wishes. It can lead to:

- Users can be forcibly added to groups without their permission
- Even after leaving a group, users can be immediately re-added
- There's no way for users to permanently opt-out or block group membership
- This could enable harassment by repeatedly adding users to undesired groups

```
function addMember(
        address account,
        KeyValue[] calldata customParams,
        RuleProcessingParams[] calldata ruleProcessingParams
    ) external override {
        uint256 membershipId = Core._grantMembership(account);
        if (_amountOfRules(IGroupRule.processAddition.selector) != 0) {
            _processMemberAddition
              (msg.sender, account, customParams, ruleProcessingParams);
        } else {
            _requireAccess(msg.sender, PID__ADD_MEMBER);
        }
        ...
    }
```

Recommendations

- Implement an acceptance mechanism where users must approve group membership before being added
- Add a blocklist allowing users to permanently block specific groups from adding them

[M-09] Potential bypass of edit post rule validation for quote posts

Severity

Impact: Medium

Likelihood: Medium

Description

When editing a post that quotes/replies to/reposts another post, the process enforces checks against the root post's rules.

```
// File: contracts/core/primitives/feed/Feed.sol

function editPost(
    uint256 postId,
    EditPostParams calldata postParams,
    KeyValue[] memory customParams,
    RuleProcessingParams[] memory feedRulesParams,
    RuleProcessingParams[] memory rootPostRulesParams,
    RuleProcessingParams[] memory quotedPostRulesParams
) external virtual override {
    --- SNIPPED ---
    uint256 quotedPostId = Core.$storage().posts[postId].quotedPostId;
    if (quotedPostId != 0) {
@1>      uint256 rootOfQuotedPost = Core.$storage
  ().posts[quotedPostId].rootPostId;
@>       _processPostEditingOnRootPost
  (rootOfQuotedPost, postId, postParams, customParams, quotedPostRulesParams);
    }
@>  uint256 rootPostId = Core.$storage().posts[postId].rootPostId;
    if (postId != rootPostId) {
@>       _processPostEditingOnRootPost
  (rootPostId, postId, postParams, customParams, rootPostRulesParams);
    }
    --- SNIPPED ---
}
```

However, for a post that quotes another post, if the quoted post has been deleted, the queried `rootPostId` will return `0` (`@1>`). This causes the protocol to check against the feed's rules instead of the original post's rules.

This behavior could lead to a bypass of rule validation, as the protocol does not revert when it encounters these specific conditions.

If `quotedPostId` is deleted, the following sequence occurs:

1. The queried `rootPostId` is returned `0`.
2. The rules is checks against feed's rules(`$feedRulesStorage()`) instead of the post's rules for `IPostRule.processEditPost` selector.
3. Since the `feedRulesStorage` does not contain the rule configuration for the `IPostRule.processEditPost` selector, the checks is skipped.

```
// File: contracts/core/primitives/feed/RuleBasedFeed.sol

function _processPostEditingOnRootPost(
    uint256 rootPostId,
    uint256 postId,
    EditPostParams memory postParams,
    KeyValue[] memory primitiveCustomParams,
    RuleProcessingParams[] memory postRulesParams
) internal {
    _processPostEditing(
        _encodeAndCallProcessEditPostOnRootPost,
        ProcessPostEditingParams({
@>          ruleSelector: IPostRule.processEditPost.selector,
@>          rootPostId: rootPostId,
            //@audit rootPostId: 0, $feedRulesStorage
            //() being used against `IPostRule.processEditPost.selector`
            postId: postId,
            postParams: postParams,
            primitiveCustomParams: primitiveCustomParams,
            rulesProcessingParams: postRulesParams
        })
    );
}
```

Recommendations

Consider explicitly tracking the root of the quoted post within the post entity, allowing reference to the rules to be queried even if the quoted post is deleted.

[M-10] Lack of upgrade-aware patterns

Severity

Impact: Medium

Likelihood: Medium

Description

The `Account`, `App`, and `Namespace` contracts inherit OpenZeppelin's `Ownable`, `base/BaseSource`, and introduce `_idToUsername` at a standard storage slot, respectively, introducing risks when upgrading to an upgrade-aware implementation due to potential storage slot mismatches.

For example, in the `Account` contract, if the contract is upgraded to use `access/Ownable` or OpenZeppelin's `OwnableUpgradeable`, the `_owner` address will no longer be accessible, leading to the `Account` contract being in a dangling state.

Recommendations

Adopt protocol self-implementation `access/Ownable` or OpenZeppelin's `OwnableUpgradeable` for `Account`, and use Namespace storage computed slot for `BaseSource` and `Namespace` to ensure proper storage slot alignment, prevent conflicts, and support future upgrades.

[M-11] Source is not reset after primitive entity removal

Severity

Impact: Medium

Likelihood: Medium

Description

The `source` field (`DATA__SOURCE`) is not reset when an entity is removed, allowing unintended reuse of the entity without requiring new verification of the action to the source.

Since the `source` is mainly used in authorizing operations from user interactions through the App, when it is already stamped to the entity and has not been removed upon entity deletion, users can potentially bypass source verification when recreating the same entity ID.

The primitives that can regenerate the same entity ID:

- **Namespace**
- **Graph** (if the claim of the dangled `followId` is available for further implementation versions).

Consider the scenario where the App (source) enables verification:

1. The user requests to create a username in the App.
2. The process first checks if the user has approved the operation and requests a signature.

3. The signature is processed on-chain with the source verification and is stamped with that username.

4. Later, the username is removed, but the source remains linked to that entity.

5. Another user (without credentials for the App) directly creates the username again with empty source custom parameters, bypassing the checks and ultimately obtaining the previous link to this entity.

Recommendations

Consider properly resetting the `source` when an entity is removed.

8.3. Low Findings

[L-01] A wrong event is emitted even if the role is already assigned to the account

`RoleBasedAccessControl.setAccess` does not check if the role has already been assigned to the account. If that is the case, the `Lens_AccessControl_AccessUpdated` event is emitted, but the state of the contract remains unchanged.

Consider adding a check to revert or, at least, not emit the event if the role is already assigned to the account.

[L-02] `BaseSource` does not allow signature cancellation

`BaseSource` contract doesn't provide a way for signers to cancel their signatures. While the `validateSource` function can be called by anyone and be used to cancel the nonce of a signature, it is expected that contracts implementing `BaseSource` will restrict access to this function to the contracts that want to validate the source.

It is recommended to implement a specific function that allows the signer to cancel their signatures by using the nonce of the signature.

[L-03] Last account's follow ID is not reusable

The `GraphCore._follow()` function allows for the reuse of follow IDs that have already been assigned to a user.

```
if (followId == 0) {
            followId = ++$storage().lastFollowIdAssigned[accountToFollow];
        } else {
@>          require(followId < $storage
//().lastFollowIdAssigned[accountToFollow], Errors.InvalidParameter()); // Only previo
            require($storage().followers[accountToFollow][followId] == address
            //(0), Errors.AlreadyExists()); // Follow ID is already taken
        }
```

However, it does not allow for the reuse of the last follow ID assigned to a user, even in case it was currently available. This is because the `require` statement uses `<` instead of `<=`.

[L-04] `changePostRules()` does not check if the post is a root post

In the `Feed` contract, the `createPost()` function only allows the author to add rules if the `postId` is equal to the `rootPostId`. This prevents the author from adding rules to a post that is a reply or a repost.

```
if (postId != rootPostId) {
        require(postParams.ruleChanges.length == 0, Errors.CannotHaveRules());
        // This covers the Reply or Repost cases
        _processPostCreationOnRootPost
          (rootPostId, postId, postParams, customParams, rootPostRulesParams);
    } else {
        _addPostRulesAtCreation(postId, postParams, feedRulesParams);
    }
```

However, the `changePostRules()` function misses this check, allowing the author to add rules to a post that is a reply or a repost.

```
function _beforeChangeEntityRules
      (uint256 entityId, RuleChange[] memory /* ruleChanges */ )
        internal
        view
        virtual
        override
    {
        require(msg.sender == Core.$storage
          ().posts[entityId].author, Errors.InvalidMsgSender());
+       require(entityId == Core.$storage
+ ().posts[entityId].rootPostId, Errors.CannotHaveRules());
    }
```

[L-05] Inconsistent return value in `_createPost()`

In the `FeedCore._createPost()` function, the return value includes `postSequentialId`, while the argument indicates `authorPostSequentialId`.

```
//File: contracts/core/primitives/feed/FeedCore.sol

function _createPost(CreatePostParams memory postParams) internal returns
  (uint256, uint256, uint256) {
    uint256 postSequentialId = ++$storage().postCount;
    --- SNIPPED ---
@>  return (postId, postSequentialId, rootPostId);
}
```

This potentially results in a mismatch between the intended and returned values in `FeedCore._createPost()`.

```
//File: contracts/core/primitives/feed/Feed.sol

function createPost(
    ...
) external virtual override returns (uint256) {
    require(msg.sender == postParams.author, Errors.InvalidMsgSender());
@>  (
  uint256postId,
  uint256authorPostSequentialId,
  uint256rootPostId
) = Core._createPost(postParams

    --- SNIPPED ---

    emit Lens_Feed_PostCreated(
        postId,
        postParams.author,
@>      authorPostSequentialId,
        ...
    );
    --- SNIPPED ---
}
```

Consider updating the `FeedCore._createPost()` function to return the correct value.