Lens V3 Extensions 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 Extensions

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 Extensions contain the bespoke implementations of contracts for Lens Dashboard and initial version of Lens Social Protocol. These contracts are opinionated and are designed to achieve the best experience while using the Lens Dashboard. These also serve as examples of how developers could build on top of the Core contracts.

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:

- `OwnerAdminOnlyAccessControl`
- `PermissionlessAccessControl`
- `Account`
- `IAccount`
- `ActionHub`
- `AccessControlFactory`
- `AccountFactory`
- `AppFactory`
- `FeedFactory`
- `GraphFactory`
- `GroupFactory`
- `LensFactory`
- `NamespaceFactory`
- `PrimitiveFactory`
- `App`
- `AppCore`
- `IApp`

7. Executive Summary

Over the course of the security review, Shaka, ubermensch, ast3ros, merlinboii engaged with Avara to review Lens V3 Extensions. In this period of time a total of **12** issues were uncovered.

Protocol Summary

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

Findings Count

Severity	**Amount**
High	3
Medium	4
Low	5
Total Findings	**12**

Summary of Findings

ID	Title	Severity	Status
[H-01]	DoS attack on App primitives using source stamp	High	Resolved
[H-02]	increaseAllowance is not included in the list of transfer-related selectors	High	Resolved
[H-03]	LensFactory does not transfer ownership of ProxyAdmin	High	Resolved
[M-01]	Metadata URI not set to source on account initialization	Medium	Resolved
[M-02]	Incorrect permission check in App.setPaymaster() function	Medium	Resolved
[M-03]	Account cannot receive ERC1155 tokens	Medium	Resolved
[M-04]	Lack of upgrade-aware patterns	Medium	Resolved
[L-01]	App.getDefaultGroup() returns always address(0)	Low	Resolved
[L-02]	Requiring canTransferNative permission even when sending funds to the account	Low	Resolved
[L-03]	Incorrect check for Accounts manager existence	Low	Resolved
[L-04]	Redundant event emissions in Account contract	Low	Resolved
[L-05]	Integer overflow in array index due to unsafe uint8 cast	Low	Resolved

8. Findings

8.1. High Findings

[H-01] DoS attack on `App` primitives using source stamp

Severity

Impact: Medium

Likelihood: High

Description

The `App` contract extends `BaseSource` abstract contract, which includes the `validateSource` function. This function validates the source stamp signature and cancels the nonce of the signature and anyone can call the function, which can lead to a DoS attack.

Imagine the following scenario:

- Alice calls the `createPost` function in a feed of the `App` contract, passing a source stamp in the `customParams` array.
- Bob sees the transaction in the mempool and frontruns it calling `App.validateSource` with the same source stamp.
- Alice's transaction fails because the nonce was already used by Bob's transaction.

Recommendations

Add a check in the `_validateSource` function override of the `App` contract to ensure that the `msg.sender` is a registered primitive contract.

[H-02] `increaseAllowance` is not included in the list of transfer-related selectors

Severity

Impact: High

Likelihood: Medium

Description

The `Account` contract requires specific permission for an account manager to execute token transfer-related actions, as well as a spending timelock period to be respected.

The `_isTransferRelatedSelector` function includes the list of selectors that are considered transfer-related. However, this list does not include the `increaseAllowance(address, uint256)` function. While this function is not part of the ERC20 standard, it is included in OpenZeppelin's ERC20 contract and is commonly present in ERC20 implementations.

As a result, manager accounts can bypass the checks on ERC20 token transfers by executing the `increaseAllowance` function.

Recommendations

Add the `increaseAllowance(address, uint256)` selector to the list of transfer-related selectors.

[H-03] `LensFactory` does not transfer ownership of `ProxyAdmin`

Severity

Impact: Medium

Likelihood: High

Description

The `LensFactory.createAccountWithUsernameFree()` function deploys a new account setting itself as the owner. This is done so that the initial setup of the account can be done, and the ownership of the account is later transferred to the `accountParams.owner` address.

```
File: LensFactory.sol

    function createAccountWithUsernameFree(
        address namespacePrimitiveAddress,
        CreateAccountParams calldata accountParams,
        CreateUsernameParams calldata usernameParams
    ) external returns (address) {
        address account = ACCOUNT_FACTORY.deployAccount(
@>          address(this),
            accountParams.metadataURI,
(...)
        IAccount(payable(account)).executeTransaction
          (namespacePrimitiveAddress, uint256(0), txData);
@>      IOwnable(account).transferOwnership(accountParams.owner);
        return account;
    }
```

However, the `owner` parameter of the `deployAccount()` function is also used to set the owner of the `ProxyAdmin` contract and this ownership is not transferred to the `accountParams.owner` address.

```
File: AccountFactory.sol

    function deployAccount(
@>      address owner,
        string calldata metadataURI,
        address[] calldata accountManagers,
        AccountManagerPermissions[] calldata accountManagersPermissions,
        SourceStamp calldata sourceStamp,
        KeyValue[] calldata extraData
    ) external returns (address) {
@>      address proxyAdmin = address(new ProxyAdmin
//(owner, _lock)); // TODO: Owner of Proxy Admin same as owner of Account
```

As a result, the account owner will not be able to perform the administrative actions over the `BeaconProxy` contract, losing for example the ability to change the implementation of the `Account` at will.

Proof of Concept

```
function test_proxyAdminOwnerIsLensFactory() public {
        address owner = address(0xbeefbeef);

        CreateAccountParams memory accountParams = CreateAccountParams({
            metadataURI: "someMetadataURI",
            owner: owner,
            accountManagers: _emptyAddressArray(),
            accountManagersPermissions: new AccountManagerPermissions[](0),
            accountCreationSourceStamp: _emptySourceStamp(),
            accountExtraData: _emptyKeyValueArray()
        });
        CreateUsernameParams memory usernameParams = CreateUsernameParams({
            username: "myTestUsername",
            createUsernameCustomParams: _emptyKeyValueArray(),
            createUsernameRuleProcessingParams: _emptyRuleProcessingParamsArray
              (),
            assignUsernameCustomParams: _emptyKeyValueArray(),

            assignRuleProcessingParams: _emptyRuleProcessingParamsArray(),
            usernameExtraData: _emptyKeyValueArray()
        });
        vm.recordLogs();
        address account = lensFactory.createAccountWithUsernameFree({
            accountParams: accountParams,
            namespacePrimitiveAddress: address(namespace),
            usernameParams: usernameParams
        });
        Vm.Log[] memory entries = vm.getRecordedLogs();
        address proxyAdmin;
        for (uint256 i = 0; i < entries.length; i++) {
            if (entries[i].topics[0] == keccak256("AdminChanged
              (address,address)")) {
                (, proxyAdmin) = abi.decode(entries[i].data,
                  (address, address));
                break;
            }
        }

        assertEq(owner, Ownable(account).owner());
        assertEq(address(lensFactory), Ownable(proxyAdmin).owner());
    }
```

Recommendations

Transfer ownership of the `ProxyAdmin` contract to `accountParams.owner` in the `createAccountWithUsernameFree()` function.

8.2. Medium Findings

[M-01] Metadata URI not set to source on account initialization

Severity

Impact: Low

Likelihood: High

Description

On account initialization, if a `SourceStamp` is provided, the source is validated but is not used to set the metadata URI, so it makes no difference sending it. This is in contrast with the `setMetadataURI` function, where the source, if present, is used to set the metadata URI.

```
function _initialize(
        string memory metadataURI,
        address[] memory accountManagers,
        AccountManagerPermissions[] memory accountManagerPermissions,
@>      SourceStamp memory sourceStamp,
        KeyValue[] memory extraData
    ) internal {
        if (sourceStamp.source != address(0)) {
@>          ISource(sourceStamp.source).validateSource(sourceStamp);
        }
(...)
@>      _setMetadataURI(metadataURI);
        emit Events.Lens_Contract_Deployed
          ("account", "lens.account", "account", "lens.account");
    }

(...)

    function setMetadataURI(
      stringcalldatametadataURI,
      SourceStampcalldatasourceStamp
    ) external override {
(...)
        if (sourceStamp.source != address(0)) {
            ISource(sourceStamp.source).validateSource(sourceStamp);
@>          _setMetadataURI(metadataURI, sourceStamp.source);
            emit Lens_Account_MetadataURISet(metadataURI, sourceStamp.source);
        } else {
            _setMetadataURI(metadataURI);
            emit Lens_Account_MetadataURISet(metadataURI, address(this));
        }
```

As a result, when the account is initialized with a `SourceStamp`, the metadata URI will be incorrectly set to `metadataURI[address(this)]` instead of `metadataURI[sourceStamp.source]`.

Recommendations

```
if (sourceStamp.source != address(0)) {
            ISource(sourceStamp.source).validateSource(sourceStamp);
+           _setMetadataURI(metadataURI, sourceStamp.source);
+           emit Lens_Account_MetadataURISet
+ (metadataURI, sourceStamp.source);
+       } else {
+           _setMetadataURI(metadataURI);
+           emit Lens_Account_MetadataURISet(metadataURI, address(this));
        }
        for (uint256 i = 0; i < accountManagers.length; i++) {
            $storage
              ().accountManagerPermissions[accountManagers[i]] = accountManagerPermiss
            emit Lens_Account_AccountManagerAdded
              (accountManagers[i], accountManagerPermissions[i]);
        }
        _decodeAndSetExtraData(extraData);
-       _setMetadataURI(metadataURI);
```

[M-02] Incorrect permission check in `App.setPaymaster()` function

Severity

Impact: Medium

Likelihood: Medium

Description

The `setPaymaster` function in the `App` contract is supposed to be accessible only with the `SET_PAYMASTER` permission. However, the function incorrectly checks for the `SET_PRIMITIVES` permission instead.

This means that a user with the `SET_PRIMITIVES` permission can set the paymaster, which is not the intended behavior. Additionally, users with the `SET_PAYMASTER` permission will not be able to call the function, and they must be granted the `SET_PRIMITIVES` permission instead, which might not be desirable.

Recommendations

```
function setPaymaster(address paymaster) external override {
-       _requireAccess(msg.sender, PID__SET_PRIMITIVES);
+       _requireAccess(msg.sender, PID__SET_PAYMASTER);
        _setPaymaster(paymaster);
    }
```

[M-03] `Account` cannot receive ERC1155 tokens

Severity

Impact: Low

Likelihood: High

Description

The `Account` contract is meant to be capable of holding ERC1155 tokens. According to EIP-1155:

> Smart contracts MUST implement all the functions in the ERC1155TokenReceiver interface to accept transfers.
>
> Smart contracts MUST implement the ERC-165 supportsInterface function and signify support for the ERC1155TokenReceiver interface to accept transfers.

However, `Account` does not implement these functions, which means that it cannot receive ERC1155 tokens.

Recommendations

Implement the `ERC1155TokenReceiver` and `ERC165` interfaces in the `Account` contract.

[M-04] 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.

8.3. Low Findings

[L-01] `App.getDefaultGroup()` returns always `address(0)`

The `App` contract implements the `getDefaultGroup()` function. However, the default group is never set, so this function always returns `address(0)`.

If the intention is not to have a default group, consider removing this function to avoid confusion.

If the intention is to have a default group, create a setter function to set the default group address.

[L-02] Requiring `canTransferNative` permission even when sending funds to the account

In order to transfer native tokens from the `Account` account contract, a manager is required to have assigned the `canTransferNative` permission. The purpose of the permission is most likely to limit managers' ability to spend the account's native tokens.

However, this restriction also prevents a manager from spending the funds sent by themselves to the `executeTransaction` function.

Consider enforcing the check for the `canTransferNative` permission only when the manager is trying to spend funds from the account's balance.

```
-       if (value > 0) {
+       if (value > msg.value) {
            require($storage(
              $storage

            ).accountManagerPermissions[msg.sender].canTransferNative, Errors.NotAllow
        }
```

[L-03] Incorrect check for `Account`s manager existence

In the `Account` contract, the `addAccountManager()`, `removeAccountManager()`, and `updateAccountManagerPermissions()` use the `canExecuteTransactions` permission to determine if a manager exists or not. Additionally, `updateAccountManagerPermissions()` enforces that the `canExecuteTransactions` is true in the new permissions. However, managers can be created without the `canExecuteTransactions` permission being set to true. There can be a manager that only has the `canSetMetadataURI` permission, so trying to remove or update the permissions of this manager will fail because the `canExecuteTransactions` permission is not set to true.

Consider checking as well if the manager has the `canSetMetadataURI` permission in order to determine if the manager exists or is a new valid permission.

[L-04] Redundant event emissions in `Account` contract

When the metadata URI is updated on the `Account` contract both the `Lens_Account_MetadataURISet(string,source)` and `Lens_Account_MetadataURISet(string)` events are emitted, which is redundant.

Similarly, when the ownership is transferred, both the `OwnershipTransferred(address,address)` and `Lens_Account_OwnershipTransferred(address)` events are emitted.

Consider removing the redundant events.

[L-05] Integer overflow in array index due to unsafe uint8 cast

The `_add` function in AppCore unsafely casts array lengths to uint8 when storing element indices, which could lead to index overflow and data

corruption if array sizes exceed 255 elements. It would cause incorrect index assignments for elements beyond index 255 and data integrity issues when removing or accessing elements.

```
function _add(address element, address[] storage array, mapping
      (address => ArrayStorageHelper) storage arrayHelper)
        internal
    {
        ...
        array.push(element);
        arrayHelper[element] = ArrayStorageHelper({index: uint8
          (array.length - 1), isSet: true});
    }
```

It's recommended to safe cast the index or add explicit bounds checking `array.length <= 256`.