SMART CONTRACT AUDIT REPORT

for

Lens Protocol

Prepared By: Yiqun Chen

Hangzhou, China

January 28, 2022

Document Properties

Client	Aave
Title	Smart Contract Audit Report
Target	Lens Protocol
Version	1.0
Author	Shulin Bie
Auditors	Shulin Bie, Xuxian Jiang
Reviewed by	Yiqun Chen
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	January 28, 2022	Shulin Bie	Final Release
1.0-rc	January 27, 2022	Shulin Bie	Release Candidate #2
1.0-rc	January 10, 2022	Shulin Bie	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Yiqun Chen
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the `Lens Protocol`, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Lens Protocol

`Lens Protocol` is a composable social graph protocol built to be community-owned and ever-evolving. It empowers its users by allowing them to decide how they want their social graph to be built, and how they want it to be monetized, if at all. Furthermore, the protocol is engineered with the concept of modularity at its core, allowing for an infinitely expanding amount of use cases. This, from the user's perspective, translates to a new paradigm of ownership and customization that just isn't possible (or financially feasible) in web2.

Table 1.1: Basic Information of Lens Protocol

Item	Description
Target	Lens Protocol
Website	https://lens.dev/
Type	Solidity Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	January 28, 2022

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/aave/lens-protocol.git (dd137b2)

And this is the Git repository and commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/aave/lens-protocol.git (52abf8d)

1.2 About PeckShield

PeckShield Inc. [10] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [9]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [8], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 3.1 to classify our findings.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the `Lens Protocol` implementation. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	2	■■
Low	0	
Informational	4	■■■■
Total	6	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 2 medium-severity vulnerabilities, and 4 informational recommendations.

Table 2.1: Key Lens Protocol Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Approved Addresses Management In ApprovalFollowModule	Coding Practices	Confirmed
PVE-002	Informational	Improved Logic Of FollowNFT::_moveDelegate()	Coding Practices	Confirmed
PVE-003	Informational	Inconsistency Between Implementation And Document	Coding Practices	Fixed
PVE-004	Informational	Suggested Fixed Compiler Version	Coding Practices	Confirmed
PVE-005	Medium	Trust Issue Of Admin Keys	Security Features	Confirmed
PVE-006	Medium	Improved Follow Logic In InteractionLogic::follow()	Business Logic	Fixed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Approved Addresses Management In ApprovalFollowModule

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `ApprovalFollowModule`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

The `Lens Protocol` implements a decentralized social media, which is achieved by allowing users to create profiles and interact with each other via these profiles. Moreover, the protocol has a modular design with three types of modules supported so far: `follow` (taking effect when a profile is followed), `collect` (taking effect when a publication is collected), and `reference` (taking effect when a publication is referred). When the user creates the profile, he can select a whitelisted `follow` module. While examining a whitelisted `follow` module `ApprovalFollowModule`, we observe the built-in management of approved addresses can be improved.

To elaborate, we show below the related code snippet of the `ApprovalFollowModule` contract. In this contract, we notice it uses the `mapping(address => mapping(uint256 => mapping(address => bool)))internal _approvedByProfileByOwner` (line 20) to record the approved addresses. If the owner of the profile wants to reinitialize the approved addresses with the call to `initializeFollowModule()` routine, the previous set of approved addresses still takes effect. Given this, we suggest to use `EnumerableSet` to manage the set of approved addresses. By doing so, the set of approved addresses are more friendly to be managed.

```
18  contract ApprovalFollowModule is IFollowModule, FollowValidatorFollowModuleBase {
19      // We use a triple nested mapping so that, on profile transfer, the previous
            approved address list is invalid;
20      mapping(address => mapping(uint256 => mapping(address => bool)))
21          internal _approvedByProfileByOwner;
22
```

```
        ...

        function initializeFollowModule(uint256 profileId, bytes calldata data)
            external
            override
            onlyHub
            returns (bytes memory)
        {
            address owner = IERC721(HUB).ownerOf(profileId);

            if (data.length > 0) {
                address[] memory addresses = abi.decode(data, (address[]));
                for (uint256 i = 0; i < addresses.length; i++) {
                    _approvedByProfileByOwner[owner][profileId][addresses[i]] = true;
                }
            }
            return data;
        }

        ...
    }
```

Listing 3.1: `ApprovalFollowModule`

Recommendation Improved the approved address management mechanism.

Status The issue has been confirmed by the team.

3.2 Improved Logic Of FollowNFT::_moveDelegate()

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `FollowNFT`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

As mentioned in Section 3.1, the `Lens Protocol` allows the user to follow a certain profile. By doing so, the user will receive a `follow NFT` that is unique to the followed profile. The `FollowNFT` contract follows the standard `ERC721` specification and can also be used for governance in allowing for users to cast and record the votes. Moreover, the `FollowNFT` contract allows for dynamic delegation of a voter to another, though the delegation is not transitive. In particular, one internal routine, i.e., `_moveDelegate()`, is called when the delegator intends to delegate his votes to the delegatee. While examining its logic, we notice its current implementation can be improved.

To elaborate, we show below the related code snippet of the `FollowNFT` contract. In the `_moveDelegate()` function, the internal `_writeSnapshot()` is executed (lines 194 and 205) to update the votes of the delegator (specified by the input `from` parameter) and delegatee (specified by the input `to` parameter). However, we notice it has not taken into account the special case where the input `from` parameter is equal to the input `to` parameter. In this case, there is no need to update the votes of the delegator and delegatee.

```
182    function _moveDelegate(
183        address from,
184        address to,
185        uint256 amount
186    ) internal {
187        // NOTE: Since we start with no delegate, this condition is only fulfilled if a delegation occurred
188        if (from != address(0)) {
189            uint256 previous = 0;
190            uint256 fromSnapshotCount = _snapshotCount[from];
191
192            previous = _snapshots[from][fromSnapshotCount - 1].value;
193
194            _writeSnapshot(from, uint128(previous - amount), fromSnapshotCount);
195            emit Events.FollowNFTDelegatedPowerChanged(from, previous - amount, block.timestamp);
196        }
197
198        if (to != address(0)) {
199            uint256 previous = 0;
200            uint256 toSnapshotCount = _snapshotCount[to];
201
202            if (toSnapshotCount != 0) {
203                previous = _snapshots[to][toSnapshotCount - 1].value;
204            }
205            _writeSnapshot(to, uint128(previous + amount), toSnapshotCount);
206            emit Events.FollowNFTDelegatedPowerChanged(to, previous + amount, block.timestamp);
207        }
208    }
```

Listing 3.2: `FollowNFT::_moveDelegate()`

Recommendation Improve the implementation of the `_moveDelegate()` routine as above-mentioned.

Status The issue has been confirmed by the team. The team decides to leave it as is since it does not pose any security risk.

3.3 Inconsistency Between Implementation And Document

- ID: PVE-003
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: LensHub
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

In the Lens Protocol, the LensHub contract is the main entry for interaction with users, which provides a series of query routines for the user. In particular, one routine, i.e., getContentURI(), is designed to retrieve the content URI mapped to the given publication. While examining its implementation and the Lens Protocol specification, we notice the description in the specification is inconsistent with its implementation.

To elaborate, we show below the related code snippet of the LensHub contract. The Lens Protocol specification specifies that "This function returns the content URI mapped to the given publication, if any. Note that content URIs only exist for posts and comments, and should always be empty for mirrors." However, in the getContentURI() function, we note that the original mirrored publication's URI rather than empty is returned when the type of the queried publication is mirror, which is inconsistent with the description in the specification.

```
    function getContentURI(uint256 profileId, uint256 pubId)
        external
        view
        override
        returns (string memory)
    {
        (uint256 rootProfileId, uint256 rootPubId, ) = Helpers.getPointedIfMirror(
            profileId,
            pubId,
            _pubByIdByProfile
        );
        return _pubByIdByProfile[rootProfileId][rootPubId].contentURI;
    }
```

Listing 3.3: LensHub::getContentURI()

Recommendation Ensure the consistency between documents (including embedded comments) and implementation.

Status The Lens Protocol specification has been updated.

3.4 Suggested Fixed Compiler Version

- ID: PVE-004
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Multiple Contracts`
- Category: Coding Practices [6]
- CWE subcategory: CWE-487 [3]

Description

Solidity releases move relatively fast, and new versions may introduce breaking changes. Often times, we have experienced the pain of switching among different versions: what might work under one version of Solidity might not work with another one. It is not unusual that we may want to start new projects running the newly released Solidity version, but we need to import third-part libraries that still use old version, say 0.4. Switching between these versions is often quite an annoying process.

Due to the fact that compiler upgrades might bring unexpected incompatibility or inter-version inconsistencies, we always suggest using fixed compiler version whenever possible. As an example, we highly encourage to explicitly indicate the Solidity compiler version, e.g., `pragma solidity 0.8.10;` instead of `pragma solidity ^0.8.0;`.

Our analysis with current Solidity use in `Lens Protocol` shows that multiple versions are simultaneously used. This is certainly allowed and may be quite common in practice. However, the simultaneous use of multiple versions will likely bring additional project-wide management and maintenance overhead. If at all possible, choose a specific compiler version and make a consistent use.

Table 3.1: Simultaneous Use of Multiple Solidity Compiler Versions

File	Compiler Version
./contracts/core/base/ERC721.sol	^0.8.0
./contracts/core/base/ERC721Enumerable.sol	^0.8.0
./contracts/core/base/LensMultiState.sol	0.8.10
./contracts/core/base/LensNFTBase.sol	0.8.10
other contracts	0.8.10

Recommendation Be consistent on using a specific compiler version.

Status The issue has been confirmed by the team. The team decides to leave it as is considering it has no exploited security vulnerability.

3.5 Trust Issue Of Admin Keys

- ID: PVE-005
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `Multiple Contracts`
- Category: Security Features [5]
- CWE subcategory: CWE-287 [2]

Description

In the `Lens Protocol`, there is a privileged account that plays a critical role in governing and regulating the protocol-wide operations (e.g., configuring various system parameters). In the following, we show the representative functions potentially affected by the privilege of the account.

```
    function setState(DataTypes.ProtocolState newState) external override {
        if (msg.sender != _governance && msg.sender != _emergencyAdmin)
            revert Errors.NotGovernanceOrEmergencyAdmin();
        _setState(newState);
    }

    /// @inheritdoc ILensHub
    function whitelistFollowModule(address followModule, bool whitelist) external
        override onlyGov {
        _followModuleWhitelisted[followModule] = whitelist;
        emit Events.FollowModuleWhitelisted(followModule, whitelist, block.timestamp);
    }

    /// @inheritdoc ILensHub
    function whitelistReferenceModule(address referenceModule, bool whitelist)
        external
        override
        onlyGov
    {
        _referenceModuleWhitelisted[referenceModule] = whitelist;
        emit Events.ReferenceModuleWhitelisted(referenceModule, whitelist, block.
            timestamp);
    }

    /// @inheritdoc ILensHub
    function whitelistCollectModule(address collectModule, bool whitelist)
        external
        override
        onlyGov
    {
        _collectModuleWhitelisted[collectModule] = whitelist;
        emit Events.CollectModuleWhitelisted(collectModule, whitelist, block.timestamp);
    }
```

Listing 3.4: `LensHub`

We emphasize that the privilege assignment may be necessary and consistent with the protocol design. However, it would be worrisome if the privileged account is not governed by a DAO-like structure. Note that a compromised account would allow the attacker to modify a number of sensitive system parameters, which directly undermines the assumption of the Lens Protocol design.

Recommendation Promptly transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status The issue has been confirmed by the team. The team intends to introduce multi-sig and timelock mechanisms to mitigate this issue.

3.6 Improved Follow Logic In InteractionLogic::follow()

- ID: PVE-006
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: LensHub/InteractionLogic
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

As mentioned in Section 3.1, the Lens Protocol allows the user to follow a certain profile (via LensHub ::follow()). In the meantime, it also allows the owner of the profile to log out the profile by burning the NFT token that uniquely identifies the profile. While examining the current logic, we notice a specific design needs to be revisited.

To elaborate, we show below the related code snippet of the contracts. We notice the InteractionLogic ::follow() function is called inside the LensHub::follow() function. In the InteractionLogic::follow () function, the following statements are executed to ensure that the followed profile does exist: string memory handle = _profileById[profileIds[i]].handle (line 47) and if (bytes(handle).length == 0)revert Errors.TokenDoesNotExist() (line 48). However, when the profile is logged out with the calling of LensHub::burn(), the _profileById[profileIds[i]].handle is not cleared. That is to say, the user can still follow the burnt profile, which may not fit the realistic scenario.

```
    /// @inheritdoc ILensHub
    function burn(uint256 profileId) external override whenNotPaused {
        if (msg.sender != ownerOf(profileId)) revert Errors.NotProfileOwner();
        _burn(profileId);
        bytes32 handleHash = keccak256(bytes(_profileById[profileId].handle));
        _profileIdByHandleHash[handleHash] = 0;
```

```
    }

    /// ****************************************
    /// *****PROFILE INTERACTION FUNCTIONS*****
    /// ****************************************

    /// @inheritdoc ILensHub
    function follow(uint256[] calldata profileIds, bytes[] calldata datas)
        external
        override
        whenNotPaused
    {
        InteractionLogic.follow(msg.sender, profileIds, datas, FOLLOW_NFT_IMPL, _profileById);
    }
```

Listing 3.5: LensHub::burn()&&follow()

```
    function follow(
        address follower,
        uint256[] calldata profileIds,
        bytes[] calldata followModuleDatas,
        address followNFTImpl,
        mapping(uint256 => DataTypes.ProfileStruct) storage _profileById
    ) external {
        if (profileIds.length != followModuleDatas.length) revert Errors.ArrayMismatch();
        for (uint256 i = 0; i < profileIds.length; i++) {
            string memory handle = _profileById[profileIds[i]].handle;
            if (bytes(handle).length == 0) revert Errors.TokenDoesNotExist();
            address followModule = _profileById[profileIds[i]].followModule;

            address followNFT = _profileById[profileIds[i]].followNFT;

            if (followNFT == address(0)) {
                followNFT = Clones.clone(followNFTImpl);
                _profileById[profileIds[i]].followNFT = followNFT;

                bytes4 firstBytes = bytes4(bytes(handle));

                string memory followNFTName = string(
                    abi.encodePacked(handle, Constants.FOLLOW_NFT_NAME_SUFFIX)
                );
                string memory followNFTSymbol = string(
                    abi.encodePacked(firstBytes, Constants.FOLLOW_NFT_SYMBOL_SUFFIX)
                );

                IFollowNFT(followNFT).initialize(profileIds[i], followNFTName, followNFTSymbol);
                emit Events.FollowNFTDeployed(profileIds[i], followNFT, block.timestamp);
            }

```

```
            IFollowNFT(followNFT).mint(follower);

            if (followModule != address(0)) {
                IFollowModule(followModule).processFollow(
                    follower,
                    profileIds[i],
                    followModuleDatas[i]
                );
            }
        }
        emit Events.Followed(follower, profileIds, block.timestamp);
    }
```

Listing 3.6: `InteractionLogic::follow()`

Recommendation Prevent the user to follow the burnt profile.

Status The issue has been addressed in the following commit: `dd137b2`.

4 | Conclusion

In this audit, we have analyzed the `Lens Protocol` design and implementation. `Lens Protocol` is a fully composable, monetizable and decentralized social graph, which aims to empower creators to own the links between them and their community. Furthermore, the protocol is engineered with the concept of modularity at its core, allowing for an infinitely expanding amount of use cases. The current code base is well organized and those identified issues are promptly confirmed and fixed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE-487: Reliance on Package-level Scope. https://cwe.mitre.org/data/definitions/487.html.

[4] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

[5] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[6] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[7] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[8] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[9] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[10] PeckShield. PeckShield Inc. https://www.peckshield.com.