Table 1: Search criteria in the vulnerability collection process.

Keywords	improper access control,improper authorization, idor,missing authorization,incorrect authorization
CWE	cwe-639,cwe-284,cwe-285,cwe-862,cwe-863

Each vulnerability in Figure 3 necessitates a distinct object-level authorization check. These checks verify various relations between the accessing object and the requester, or between related objects, each corresponding to a distinct authorization model. To detect such vulnerabilities, we need to precisely analyze the fine-grained authorization model for each object access and then verify whether the authorization model has been properly implemented.

3 EMPIRICAL STUDY

The USPS hack is a classic example of a broken authorization vulnerability. User A was able to authenticate to the API and then pivot and access user B's and 60 million other people's information.

— Dan Barahona, Head of Marketing at Biz Dev at APIsec [9]

In this section, we first present the vulnerability collection process for our empirical study, then conduct a comprehensive study on 101 BOLA vulnerabilities. This study aims to answer the following questions: what object-level authorization models are present in real-world applications, and what are the root causes of BOLA vulnerabilities?

3.1 Vulnerability Collection and Analysis

To gather a sufficient number of BOLA vulnerabilities, we initially used the keywords and CWE types from Table 1 to search the CVE database [1] from January 2021 to December 2023. During this three-year period, we collected a total of 2,736 CVEs. We then extended our search to the bug bounty platform Huntr [69], using the same keywords from Table 1. This approach allowed us to identify 118 additional vulnerabilities not documented in the CVE database, resulting in a total dataset of 2,854 vulnerabilities.

For the purpose of our study, we only consider BOLA vulnerabilities from open-source applications that have valid patches. Consequently, we filtered out 2,482 vulnerabilities without links to corresponding source code patches in their reports . We then manually examined the remaining 372 vulnerabilities, identifying 101 BOLA vulnerabilities.

Next, each vulnerability was analyzed individually by three authors, who annotated the authorization model, root cause, and fix for each vulnerability. After this annotation process, all the involved authors discussed the annotations for each vulnerability together to form a unified terminology. In cases of disagreement, a fourth author intervened in the discussion until a consensus was reached. This collection and analysis process took 3 authors 2 months.

3.2 Caveats

While our study of vulnerabilities is thorough, it cannot exhaustively cover all BOLA vulnerabilities, and thus some results may not be universally applicable. To mitigate this issue, we have included BOLA vulnerabilities both with and without CVE IDs.

Acknowledging the possibility of bias due to the manual nature of our research, we minimized this bias by having at least three authors

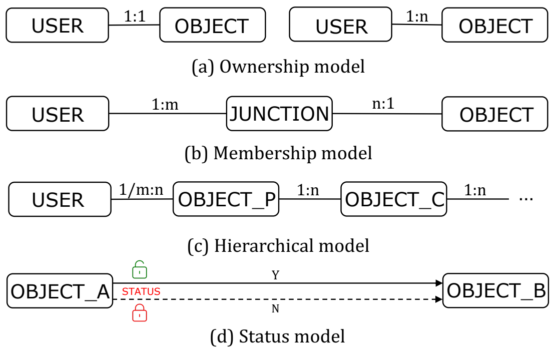

Figure 4: Four different object-level authorization models.

independently verify each vulnerability, following a methodology established in earlier studies [5, 19].

3.3 Authorization Models

Finding 1: There are four types of object-level authorization models, all of which can be derived by reasoning about relationships between tables from implicit foreign key references.

Figure 4 illustrates the four authorization models in database-backed applications. The first three models depict distinct relationships between a user and an object. The last model, named *the status model*, relates an object to the status of another object, suggesting the necessity to recognize the status column in detecting violations of this specific model.

3.3.1 Ownership model. The ownership model is perhaps the most studied authorization model, wherein a user directly owns an object as characterized by a direct one-to-one (1:1) or one-to-many (1:n) relationship between the user and the object, or more specifically, between a user table and an object table. Under this model, an object can only be manipulated by its owner. For instance, in Figure 2, a user owns their personal profile (1:1) and multiple posts created by them (1:n). Consequently, a post can only be deleted by its owner, as exemplified in Figure 3(a).

3.3.2 Membership model. In this model, the relationship between users and objects is modeled as many-to-many (m:n), indicating that (1) an object can be accessed by a specific group of users, and (2) a user can have access to multiple objects. The relationship between a `user` and a `forum` in Figure 2 is such an example: a user can participate in multiple forums, and conversely, a forum can accommodate multiple users.

To capture this membership relationship, a *junction table* (such as `user_forum` in Figure 2) is often introduced to join the user table and the object table together, establishing the connections between users and objects.

3.3.3 Hierarchical model. This model is a combination of an ownership or membership model with one or multiple parent-child relationships between objects, where a parent object can own multiple child objects. For instance, in Figure 2, a user owns their post, and a post owns all comments on itself. Thus, the owner of a post also indirectly owns all comments on that post. Consequently, only