A4-Insecure Direct Object References

Exploitability: EASY | Prevalence: COMMON | Detectability: EASY | Technical Impact: MODERATE

Description

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Attack Mechanics

If an applications uses the actual name or key of an object when generating web pages, and doesn't verify if the user is authorized for the target object, this can result in an insecure direct object reference flaw. An attacker can exploit such flaws by manipulating parameter values. Unless object references are unpredictable, it is easy for an attacker to access all available data of that type.

For example, the insure demo application uses userid as part of the url to access the allocations (/allocations/{id}). An attacker can manipulate id value and access other user's allocation information.

How Do I Prevent It?

1. **Check access:** Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.
2. **Use per user or session indirect object references:** Instead of exposing actual database keys as part of the access links, use temporary per-user indirect reference. For example, instead of using the resource's database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 or unique random numbers to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server.
3. **Testing and code analysis:** Testers can easily manipulate parameter values to detect such flaws. In addition, code analysis can quickly show whether authorization is properly verified.

Source Code Example

In `routes/allocations.js` , the insecure application takes user id from url to fetch the allocations.

```
    var userId = req.params.userId;
    allocationsDAO.getByUserId(userId, function(error, allocations) {

        if (error) return next(error);

        return res.render("allocations", allocations);
    });
```

A safer alternative is to always retrieve allocations for logged in user (using `req.session.userId`)instead of taking it from url.