A8-Cross-Site Request Forgery (CSRF)

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

Description

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests that the vulnerable application processes as legitimate requests from the victim.

Attack Mechanics

As browsers automatically send credentials like session cookies with HTTP requests to the server where cookies were received from, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones.

For example, CSRF vulnerability can be exploited on profile form on the insecure demo application.

To exploit it:

1. An attacker would need to host a forged form like below on a malicious sever.

```
    <html lang="en">
    <head></head>
        <body>
                <form method="POST" action="http://TARGET_APP_URL_HERE/profile">
                        <h1> You are about to win a brand new iPhone!</h1>
                        <h2> Click on the win button to claim it...</h2>
                        <input type="hidden" name="bankAcc" value="9999999"/>
                        <input type="hidden" name="bankRouting" value="88888888"/>
                                <input type="submit" value="Win !!!"/>
                </form>
        </body>
    </html>
```

Note: A sample app containing form for CSRF attack on NodeGoat app is available here (https://github.com/ckarande/nodegoat-csrf-attack).

2. Next, attacker would need to manage opening the form on logged in victim's browser and attract user to submit it. When user submits this form, it results in victim user's browser sending a malicious request to vulnerable server, causing CSRF attack.

How Do I Prevent It?

Express csrf middleware provides a very effective way to deal with csrf attack. By default this middleware generates a token named "_csrf" which should be added to requests which mutate state (PUT, POST, DELETE), within a hidden form field, or query-string, or header fields.

If using method-override middleware, it is very important that it is used before any middleware that needs to know the method of the request, including CSRF middleware. Otherwise an attacker can use non-state mutating methods (such as GET) to bypass the CSRF middleware checks, and use method override header to convert request to desired method.

When form is submitted, the middleware checks for existence of token and validates it by matching to the generated token for the response-request pair. If tokens do not match, it rejects the request. Thus making it really hard for an attacker to exploit CSRF.

Source Code Example

The `server.js` includes the express CSRF middleware after session is initialized. Then creates a custom middleware to generate new token using `req.csrfToken();` and exposes it to view by setting it in `res.locals`

```
        //Enable Express csrf protection
        app.use(express.csrf());

        app.use(function(req, res, next) {
            res.locals.csrftoken = req.csrfToken();
            next();
        });
```

Next, this token can be included in a hidden form field in `views/profile.html` as below.

```
    <input type="hidden" name="_csrf" value="{{ csrftoken } }">
```