Preventing Cross-Site Request Forgery (CSRF) Attacks

Cross-Site Request Forgery (CSRF)

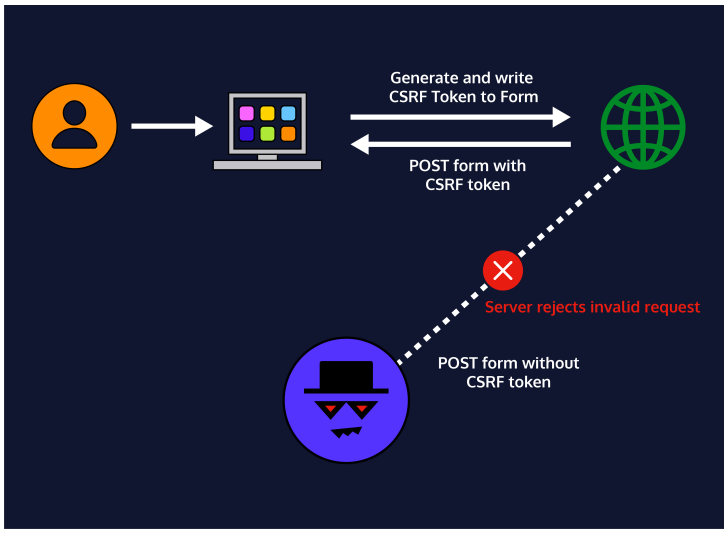
Cross-Site Request Forgery is a serious vulnerability that results from poor session management. If the requests sent by an application aren't unique, it's possible for an attacker to craft a special request and send that to a user. If the user interacts with the crafted request, and sessions aren't handled properly, an attacker may be able to assume the session identity of that user and carry out requests on their behalf.

In many cases of CSRF, a malicious actor crafts a URL embedded with a request like so:

```
http://bank.com/send?
recipient=Stranger&amount=2000
```

Preventing CSRF Attacks

Cross-Site Request Forgery (CSRF) attacks are relatively easy to mitigate. One of the simplest ways to accomplish this is through the use of CSRF tokens, which are unique values dynamically generated by a server-side application and sent to the client. Since these values are unique for every request, and constantly changing, it is nearly impossible for an attacker to pre-create the URLs/requests for an attack.



Configuring csurf

The npm package, `csurf`, protects Node.js express applications from CSRF attacks by providing middleware functions to send and process CSRF tokens with web requests. Because the CSRF tokens need to be stored in either a cookie or a session, we can configure the express app to use the `cookie-parser` module.

```
const express = require('express');
const cookieParser = require('cookie-
parser');
const csurf = require('csurf');
const app = express();

const csrfMiddleware = csurf ({
  cookie: {
    maxAge: 300000000,
    secure: true
  }
});

app.use(cookieParser);
app.use(csrfMiddleware);
```

Generating CSRF tokens with csurf

Once the `csurf` module is configured, its functions are available on all Express `get`, `post`, and `all` routes. In this code, `req.csrfToken()` generates the CSRF token which we pass to the `render()` function to allow the client's browser to access the token.

```
app.get('/form', (req, res) => {
  res.render('formTemplate',
{ csrfToken: req.csrfToken() });
});
```

It is common practice to place the CSRF token as a `hidden <input>` field within a form to submit it automatically with the contents of the form.

```
<form action="/submit" method="POST">
   <input type="hidden" name="_csrf"
value="<%= csrfToken %>" />
   <label for="body">Enter message:
</label>
   <input id="body" name="message"
type="text" />
   <input type="submit" value="Submit" />
</form>
```