Lab Exercise csp1

This is a lab exercise on developing secure software. For more information, see the introduction to the labs.

1. Task

Please change the code below so that this server-side JavaScript web application will have many security hardening headers, including a special configuration of its Content Security Policy (CSP).

2. Background

In this exercise, we'll add security headers to a server-side web application. This server-side program is written in JavaScript using the Express framework (version 4). We'll be using the `helmet` library, which helps add hardening headers to Express applications.

3. Task Information

You'll first need to load the `helmet` library. There are many ways to do this in JavaScript. In this case, we'll use a `require` statement to load a library. Look at the line below to require Express, and create a similar line to require helmet.

We now need to write code to use helmet to insert hardening headers. In Express you can use an `app.use(...)` call to indicate something to use on every request. You *could* simply call helmet like this, and it would set up many hardening headers:

```
app.use(helmet());
```

However, it's very common to need to specially configure this for your application. The helmet() call accepts an optional object that lets you configure things. In JavaScript an object is noted as `{ ... }`, where its contents are a comma-separated sequence of key-value pairs. Each key-value pair has a field name, a colon (:), and its corresponding value.

In this case we want to modify the configuration of the Content Security Policy (CSP). You can indicate this by setting the field name `contentSecurityPolicy` to a value. This value is another object. This would look like this:

```
app.use(helmet({
  contentSecurityPolicy: {
  }
}));
```

The value object for `contentSecurityPolicy` can itself have various fields, including the field `directives`. This field `directives` in turn takes a JavaScript object as its value.

The object of the `directives` field can take one or more keys. In our case we'll want to set two:

- "script-src": This will set the CSP script-src directive, which specifies valid sources for JavaScript for the client to run. Like any security setting, you want to minimize these privileges.
- "style-src": This will set the CSP style-src directive, which specifies where styles can be loaded from.

One quirk about JavaScript syntax: keys that have a dash ("-") character in it, like these, must be represented as strings (surround the term with double quotes).

In the case of "script-src", for our purposes set it to the array `["'self'", "https://example.com"]`. This means that JavaScript can be run from either this server *or* from the website `https://example.com` - and nowhere else. Note that `'self'` is itself quoted. We could have also included "unsafe-inline" which would allow the use of inline JavaScript. However, we won't do that in this example, because it's much safer to *not* include "unsafe-inline". When a CSP "script-src" value is set, and it doesn't include "unsafe-inline", that means that JavaScript embedded in some HTML will *not* be executed. This is great for security; often attackers can trick servers into slipping something into some generated HTML, and this means that if it's a script, it won't be executed.

Set "style-src" to `["'self'"]` - that states that styles can *only* come from this site (and nowhere else). Again, we didn't include "unsafe-inline", that means that CSS embedded in the HTML will be ignored. This is good for security, because it means that even if an attacker tricks a server into embedding some CSS commands, those commands will be ignored.

Use the “hint” and “give up” buttons if necessary.

4. Interactive Lab (COMPLETE!)

```
const express = require("express");
const helmet = require("helmet");

const app = express();
```

```
// Use Helmet to insert hardening headers
app.use(helmet({
 contentSecurityPolicy: {
 directives:{
 "script-src":["'self'", "https://example.com"],
 "style-src":["'self'"]
 }
 }
 }));
```

```
app.get("/", (req, res) => {
  res.send("Hello world!");
});

app.listen(3000);
```

Hint Reset Give up

This lab was developed by David A. Wheeler at The Linux Foundation.

Completed 2025-03-06T21:36:18.552Z f60e32e1-48c7-4104-b955-198377fba31f 142idch1fpxb71