

Consideraciones de Seguridad al Usar el Framework Web *Express* en Producción

Nicolás Aguado – nico@nico.eus

SGSSI — 17 de diciembre de 2024

1. Introducción

Actualmente, una de las principales herramientas para el desarrollo de aplicaciones web es el framework **Express** [12] para **Node.js** [13].

Este framework destaca por su facilidad de uso y su simpleza a la hora de diseñar aplicaciones. Esta aparente sencillez es eclipsada por todos los aspectos que hay que tener en cuenta a la hora de hacer un despliegue real, en producción.

Como punto de partida para demostrar las experimentaciones descritas a continuación, se puede encontrar un caso práctico *dockerizado* en el repositorio de github [nicoagr/vulnode](https://github.com/nicoagr/vulnode).

2. Consideraciones Básicas

De cara a no repetir código y a tener una base inicial, muchas de las aplicaciones que se diseñan usando este framework web se generan mediante la utilidad **express-generator** [10]. Esta utilidad está bien, pero hay que tener una serie de detalles en cuenta para garantizar un despliegue seguro.

- Al usar la utilidad de generación, se establecen como dependencias unas versiones de los paquetes (de **express**, el *engine* de plantillas, etc.) vulnerables. Es recomendable ejecutar el comando `npm audit fix --force` (y después revisar por si hay cambios conflictivos).
- Lo más habitual es encontrarse una configuración en la que a las aplicaciones se accede *a través* de un reverse proxy. Para ello, se recomienda establecer el parámetro **trust-proxy** adecuadamente [11].
- De cara a reducir vectores de ataque y divulgaciones de información, se recomienda establecer la variable de entorno **NODE_ENV** a **PRODUCTION**, ya que el modo de operación por defecto de **Express** es el modo desarrollo.
- Con el objetivo de mitigar vulnerabilidades web comunes, se recomienda usar las extensiones **helmet** [8], **cors** [7], **xss** [14] y **hpp** [9].
- Para proteger frente a ataques cuyo objetivo son los recursos del sistema, se recomienda utilizar un *middleware* que haga funciones de *rate-limiting* activo o pasivo, como por ejemplo **express-rate-limit** [1] y/o **express-slow-down** [2].

Estas consideraciones se encuentran implementadas en el fichero [app.js](#) en el repositorio previamente mencionado.

3. Función *eval()*

Una práctica de seguridad que se desea recalcar con especial atención es el uso de la función `eval()` para conversiones entre tipos y evaluaciones dinámicas.

Con un poco de prisa y sin prestar atención a lo que se quiere hacer, un ojo inexperto puede llegar a cierto tipo de búsquedas en foros especializados [3] [4] y utilizar `eval()` para hacer conversiones entre tipos. Esto en sí no es un problema, pero se puede llegar a convertir en uno si no se regula el input del usuario y se permiten ejecuciones directas.

Véase el siguiente ejemplo en el que se está recibiendo un input del usuario para hacer una acción y se está convirtiendo el string obtenido a tipo JSON mediante la función `eval()`.

```
routes/eval.js
router.post('/', (req, res) => {
  try {
    const parsedJson = eval(`${req.body}`);
    // do something with parsedJson
    res.status(200).send('OK');
  } catch (error) {
    res.status(400).send('Invalid input format');
  }
});
```

A pesar de todas las protecciones establecidas en la Sección 2, el mal uso de esta función puede llegar a ser peligroso si se puede llegar a hacer una petición `POST`, por ejemplo, mediante *fetch* desde la consola del navegador y de la siguiente manera para llegar a ejecutar código en la máquina de la víctima.

```
routes/eval.js
fetch('http://localhost:8888/eval/', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    fruit:
      "require('child_process').exec('touch /tmp/test')"
  })
});
```

Aquí, el comando `touch /tmp/test` puede llegar a ser reemplazado con cualquier otro comando, siendo la vulnerabilidad expuesta de tipo **Remote Code Execution** (CWE-94).

En la aplicación provista anteriormente, se pueden llegar a ejecutar comandos en el contenedor docker virtualizado haciendo la petición `POST` en la ruta `/eval`.

4. Cookies Inseguras en el Cliente

Puede haber ocasiones en las que un programador decida guardar datos de sesión y/o autenticación en las cookies del cliente (en lugar de guardarlas del lado del servidor). A priori, siempre que estas cookies estén encriptadas y/o contengan datos asociados de autenticación (tags) esta es una buena práctica.

Pero, en muchos de los casos, no se presta la atención que se debiera a la configuración de las cookies. Según un estudio [5] realizado en el 2018, el 72.58 % de los despliegues realizados usando el middleware de express `cookie-session` tenían como clave secreta de encriptación `secret key`.

Para ver cómo explotar esta vulnerabilidad, supóngase un caso en el que se está usando el citado middleware (podría ser cualquier otro) y una clave secreta insegura en una aplicación de autenticación simple (en el ejemplo concreto desplegado, en la ruta `/cookie/`). Se pueden llegar a utilizar herramientas tales como `cookie-monster` [6] para llegar a extraer la clave secreta de autenticación (el tag) por fuerza bruta e incluso iniciar sesión con otros usuarios. En la Figura 1 se puede llegar a ver un ejemplo de un *crackeo* de una cookie.

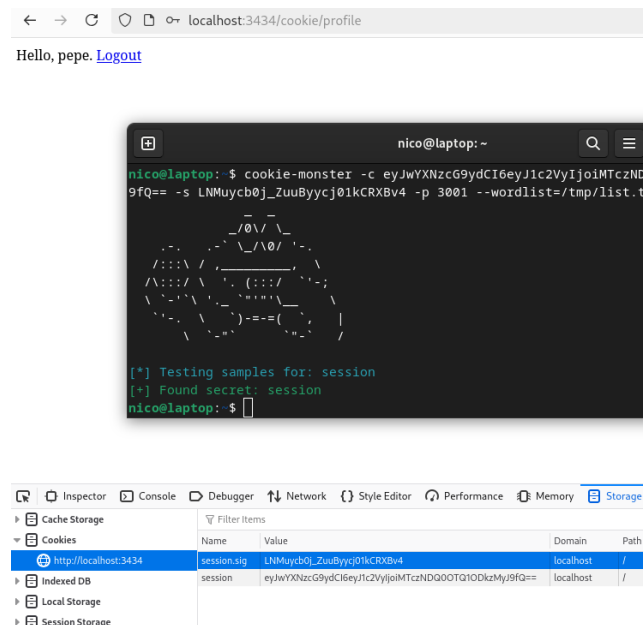


Figura 1: Ataque de fuerza bruta a una cookie del middleware `cookie-session`

Entonces, una vez se consigue la clave secreta, se puede generar una nueva firma con datos arbitrarios y manipular las cookies en una configuración *a la carta*. Por ejemplo, usando `cookie-monster`, se puede observar una codificación con la clave secreta `secret` en la Figura 2.

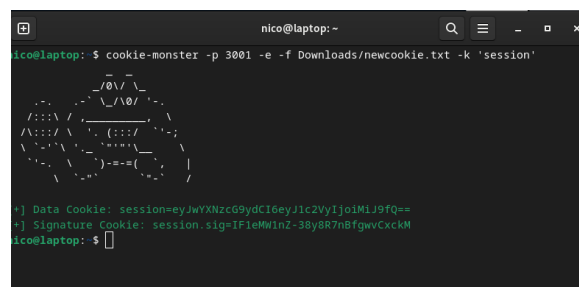


Figura 2: Generación de un tag de verificación partiendo de una clave secreta conocida

Una modificación de cookies por parte del usuario puede llegar a suponer un riesgo significativo para la seguridad, ya que permite evadir controles, violar políticas de acceso, comprometer información sensible y generar comportamientos no deseados dentro de la aplicación.

La vulnerabilidad expuesta en este ejemplo es la relativa a **Inadequate Encryption Strength** ([CWE-326](#))

En conclusión, se recalca la necesidad de leer la documentación y pararse a configurar en condiciones las distintas configuraciones que se utilicen dentro de las librerías en dentro del framework `express`.

Para más información acerca de la vulnerabilidad de manipulación de cookies, se recomienda consultar el informe realizado por DigitalInterruption [\[5\]](#).

Referencias

- [1] Basic Rate Limiting Middleware for Express. <https://www.npmjs.com/package/express-rate-limit>, [Online; consultado el 17-Dic-2024]
- [2] Basic Slow-Down Middleware for Express. <https://www.npmjs.com/package/express-slow-down>, [Online; consultado el 17-Dic-2024]
- [3] Javascript - How to return or parse an object literal. <https://stackoverflow.com/a/38524802>, [Online; consultado el 17-Dic-2024]
- [4] Javascript - JSON parse vs eval. <https://stackoverflow.com/a/1843399>, [Online; consultado el 17-Dic-2024]
- [5] DigitalInterruption: Are your Cookies telling your Fortune? <https://research.digitalinterruption.com/2018/06/04/are-your-cookies-telling-your-fortune/>, [Online; consultado el 17-Dic-2024]
- [6] DigitalInterruption: Cookie-Monster - A utility for automating the testing and re-signing of Express.js cookie secrets. <https://github.com/DigitalInterruption/cookie-monster>, [Online; consultado el 17-Dic-2024]
- [7] Goode, T.: CORS Middleware for Express or Connect. <https://www.npmjs.com/package/cors>, [Online; consultado el 17-Dic-2024]
- [8] Hann, E.: Help secure Express apps by setting HTTP response headers. <https://www.npmjs.com/package/helmet>, [Online; consultado el 17-Dic-2024]
- [9] Kamenzky, N.: Express middleware to protect against HTTP Parameter Pollution attacks. <https://www.npmjs.com/package/hpp>, [Online; consultado el 17-Dic-2024]
- [10] OpenJSFoundation: Express application generator. <https://expressjs.com/en/starter/generator.html>, [Online; consultado el 17-Dic-2024]
- [11] OpenJSFoundation: Express behind proxies. <https://expressjs.com/en/guide/behind-proxies.html>, [Online; consultado el 17-Dic-2024]
- [12] OpenJSFoundation: Express Web Application Framework. <https://expressjs.com/>, [Online; consultado el 17-Dic-2024]
- [13] OpenJSFoundation: Node-JS - Cross Platform Javascript Runtime. <https://nodejs.org/en>, [Online; consultado el 17-Dic-2024]
- [14] Zongmin, L.: Express XSS Sanitizer. <https://www.npmjs.com/package/express-xss-sanitizer>, [Online; consultado el 17-Dic-2024]