

Esplorazione dei CVE tramite AI

Obbiettivo:

L'obiettivo di questa attività è acquisire familiarità con l'uso di AI come strumento per la ricerca di informazioni sui CVE (Common Vulnerabilities and Exposures).

Prompt in input :

“Sono uno studente di cybersecurity e sto facendo uno studio sulle varie CVE di wordpress in modo da capire studiare e risolvere o dare una mano a farlo. Elenca quindi tutte le CVE conosciute fino ad oggi spiegando in maniera tecnica chiara e dettagliata che mi permettano di capire e comprendere meglio il tutto. Inoltre scrivimi anche best practice o altre cose che mi permettevano di capire a pieno tutto e formarmi come esperto di cybersecurity.”

Risposta Gemini :

Per darti una risposta onesta e tecnicamente accurata, devo farti una premessa fondamentale: **elencare "tutte" le CVE (Common Vulnerabilities and Exposures) di WordPress è impossibile in una singola risposta.**

WordPress esiste da oltre 20 anni. Esistono CVE per il **Core** (il motore principale), ma la stragrande maggioranza delle vulnerabilità risiede nei **Plugin** e nei **Temi** (l'ecosistema). Parliamo di decine di migliaia di vulnerabilità storiche.

Tuttavia, per il tuo studio, è molto più utile analizzare le **classi di vulnerabilità** più frequenti e studiare **CVE storiche significative** che fungono da caso di studio ("Case Study").

Ecco un'analisi tecnica strutturata per aiutarti a comprendere, replicare (in ambiente controllato) e risolvere queste falle.

1. Anatomia delle Vulnerabilità WordPress

Per diventare un esperto, non devi memorizzare i numeri delle CVE, ma capire i **pattern**. Ecco le categorie principali che troverai nel 90% dei casi:

A. SQL Injection (SQLi)

- **Concetto:** L'attaccante inietta comandi SQL malevoli attraverso input non sanitizzati (es. parametri GET/POST o cookie) che vengono passati direttamente al database.
- **Impatto:** Furto di dati (dump del DB), bypass dell'autenticazione, creazione di utenti admin.

B. Cross-Site Scripting (XSS)

- **Concetto:** L'attaccante inietta script JavaScript malevoli nelle pagine visualizzate da altri utenti.
- **Tipi:**
 - *Stored:* Lo script viene salvato nel DB (es. nei commenti o nei post) ed eseguito ogni volta che qualcuno visita la pagina.
 - *Reflected:* Lo script è nel link che la vittima clicca.
- **Impatto:** Furto di sessioni (cookie hijacking), redirect malevoli, defacement.

C. Remote Code Execution (RCE)

- **Concetto:** Il "Santo Graal" degli hacker. Permette di eseguire codice PHP arbitrario sul server.
- **Impatto:** Compromissione totale del server. Spesso deriva da funzionalità di upload file insicure o deserializzazione di oggetti.

D. Privilege Escalation / Auth Bypass

- **Concetto:** Un utente con bassi privilegi (es. "Subscriber") riesce a compiere azioni da amministratore o a loggarsi senza password.

2. Analisi Tecnica di CVE Significative (Case Studies)

Analizziamo alcune CVE reali per capire il codice vulnerabile e la fix.

Caso Studio 1: SQL Injection nel Core (CVE-2022-21661)

Questa è una vulnerabilità critica nel core di WordPress (versioni < 5.8.3).

- **Il problema:** La classe `WP_Query` non sanitizzava correttamente i dati quando venivano passati parametri specifici ("clean queries").
- **Dettaglio Tecnico:** La vulnerabilità risiedeva nel modo in cui venivano costruite le query SQL all'interno di `wp-includes/class-wp-tax-query.php`. Se un plugin

o un tema utilizzava `WP_Query` prendendo input direttamente dall'utente senza validazione, un attaccante poteva manipolare l'array dei termini.

Codice vulnerabile (Concettuale):

PHP

```
// Se $user_input non è validato
```

```
$query = new WP_Query( array( 'tax_query' => $user_input ) );
```

```
// Il motore interno non effettuava l'escape corretto di $user_input
```

-
- **La Soluzione (Fix):** WordPress ha aggiornato il core per forzare l'uso di `$wpdb->prepare()` anche in questi contesti specifici, assicurandosi che ogni input sia trattato come stringa letterale e non come comando SQL.

Caso Studio 2: RCE via "Bricks Builder" (CVE-2024-25600)

Una vulnerabilità recentissima e devastante in un tema/plugin premium molto diffuso.

- **Il problema:** Esecuzione di codice remoto non autenticata.
- **Dettaglio Tecnico:** Il codice utilizzava una funzione `eval()` (o simile meccanismo dinamico) su input fornito dall'utente. Il plugin aveva una funzione `prepare_query_vars_from_settings` che prendeva input JSON. L'errore era fidarsi dell'input utente e passarlo a funzioni che eseguivano codice PHP per "renderizzare" il contenuto.
- **Snippet concettuale dell'attacco:** L'attaccante inviava un payload speciale a una REST API endpoint esposta. Il payload conteneva codice PHP che il sistema eseguiva pensando fosse parte del layout del sito.
- **La Lezione:** Mai passare input utente a funzioni come `eval()`, `exec()`, `passthru()` o `call_user_func()` senza whitelist strict.

Caso Studio 3: Privilege Escalation in "Ultimate Member" (CVE-2023-3460)

- **Il problema:** Un utente non registrato poteva registrarsi come Amministratore.
- **Dettaglio Tecnico:** Il plugin permetteva agli utenti di inviare un array di dati durante la registrazione. Non c'era un controllo (whitelist) sulle chiavi dell'array. L'attaccante intercettava la richiesta di registrazione (usando Burp Suite) e aggiungeva il parametro: `wp_capabilities[administrator]=1` Poiché il codice salvava tutti i meta-dati inviati (`update_user_meta`), l'utente veniva creato con permessi da admin.
- **La Soluzione:** Definire sempre una whitelist di campi che l'utente può modificare.

3. Strumenti e Metodologia di Studio

Per studiare queste CVE, non limitarti a leggere. Crea un laboratorio.

Il tuo Lab:

1. **LocalWP o Docker:** Installa versioni vecchie di WordPress o Plugin vulnerabili (cerca su *WPDIRECTORY* o repository vecchi).
 2. **WPScan:** Impara a usare questo tool da terminale Kali Linux. `wpscan --url http://tuo-sito-locale --enumerate p` (enumera plugin vulnerabili).
 3. **Burp Suite:** Fondamentale per intercettare le richieste e modificare i payload (come nell'esempio di Privilege Escalation).
 4. **Query Monitor:** Plugin per vedere quali query SQL vengono eseguite in tempo reale (utile per debuggare SQLi).
-

4. Best Practices per lo Sviluppatore/Auditor (La via dell'Esperto)

Per diventare un esperto e "risolvere" i problemi, devi padroneggiare queste tecniche di difesa nel codice PHP:

A. Sanitizzazione, Validazione ed Escaping (La Triade Sacra)

Devi sapere esattamente quando usare cosa:

- **Validazione (Input):** Controllo i dati *appena arrivano*.
 - Esempio: `is_email()`, `absint()` (per forzare un intero).
- **Sanitizzazione (Input):** Pulisco i dati prima di *salvarli* nel DB.
 - Esempio: `sanitize_text_field()`, `sanitize_email()`.
- **Escaping (Output):** Pulisco i dati prima di *stamparli* a schermo (previene XSS).
 - Esempio: `esc_html()`, `esc_url()`, `esc_attr()`.

B. Nonces (Numbers used ONCE)

In WordPress, i Nonce proteggono contro attacchi CSRF (Cross-Site Request Forgery).

Regola d'oro: Ogni azione che modifica lo stato (salvare impostazioni, cancellare post) deve avere un controllo once.

PHP

```
// Creazione nel form
```

```
wp_nonce_field( 'mia_azione_salvataggio', 'mio_nome_nonce' );
```

```
// Verifica nel backend
```

```
if ( ! isset( $_POST['mio_nome_nonce'] ) || ! wp_verify_nonce( $_POST['mio_nome_nonce'], 'mia_azione_salvataggio' ) ) {  
    die( 'Security check failed' );  
}
```

C. Prepared Statements

Mai inserire variabili direttamente nelle query SQL.

- **Sbagliato:** `$wpdb->query("SELECT * FROM table WHERE id = $id");`

Corretto:

PHP

```
$wpdb->query(
    $wpdb->prepare(
        "SELECT * FROM table WHERE id = %d", // %d per interi, %s per stringhe
        $id
    )
);
```

-

D. Controllo delle Capabilities

Non controllare solo se l'utente è loggato (`is_user_logged_in()`), controlla se ha il *permesso* di fare quella cosa.

- Usa: `current_user_can('manage_options')` o `current_user_can('edit_posts')`.

Sintesi / Conclusione :

Il Vettore d'Attacco: La maggior parte delle vulnerabilità non risiede nel *Core* di WordPress, ma nell'ecosistema di **Plugin e Temi** di terze parti, spesso scritti con minore rigore sicurezza.

Le "Big 4" Vulnerabilità:

- **SQL Injection (SQLi):** Manipolazione del database.
- **XSS (Cross-Site Scripting):** Iniezione di script nel browser degli utenti.
- **RCE (Remote Code Execution):** Presa di controllo del server (la più critica).
- **Broken Access Control:** Utenti normali che diventano amministratori.

La Causa Radice: Quasi tutte le CVE nascono da un unico errore: **fidarsi dell'input utente**. Se un dato entra nel sistema senza essere validato e esce senza essere "escapato" (pulito), è una vulnerabilità.

La Difesa (Code Hardening):

- **Sanitize/Validate:** Pulisci i dati in ingresso.
- **Prepare:** Usa statement preparati per il database.
- **Escape:** Pulisci i dati in uscita (output).
- **Nonce:** Proteggi le azioni (form) con token di sicurezza unici.