VULNERABILITA’ VERBI HTTP

I verbi HTTP, o metodi HTTP, possono essere soggetti a varie vulnerabilità a seconda di come vengono implementati e utilizzati dal server.
Il phpMyAdmin presente sulla metasploitable presenta i seguenti verbi : GET,HEAD,POST,OPTIONS,TRACE
Il metodo GET è comunemente usato per recuperare dati da un server. Tuttavia, è suscettibile a diverse vulnerabilità, tra cui:

- **Iniezione SQL (SQL Injection)**: Se i parametri della query string (parte dell'URL dopo ?) non sono correttamente sanitizzati, un utente malintenzionato può inviare input dannoso che viene eseguito sul database.
- **Furto di Informazioni (Information Disclosure)**: I dati sensibili possono essere esposti nell'URL (come i parametri della query string), che possono essere memorizzati nei log del server, nella cronologia del browser, o condivisi accidentalmente.
- **Caching di Informazioni Sensibili**: Le risposte delle richieste GET possono essere memorizzate nella cache del browser o nei proxy intermedi. Se i dati sensibili sono inclusi nella risposta senza controlli di cache appropriati, potrebbero essere esposti ad altri utenti.

Il metodo HEAD è simile a GET ma recupera solo gli header della risposta senza il corpo. Sebbene il metodo HEAD non esponga il contenuto delle risposte come GET, può comunque essere sfruttato per:

- **Raccolta di Informazioni (Information Gathering)**: Gli attaccanti possono utilizzare HEAD per raccogliere informazioni sulla configurazione del server, gli header HTTP e le versioni dei software utilizzati senza recuperare il contenuto delle pagine web, riducendo il rischio di rilevamento.
- **Rilevamento di Risorse Sensibili**: Se il server non gestisce correttamente le risposte HEAD, potrebbe rivelare informazioni su risorse che non dovrebbero essere esposte.

Il metodo POST viene utilizzato per inviare dati al server, spesso per creare o aggiornare risorse. Le vulnerabilità comuni includono:

- **Cross-Site Request Forgery (CSRF)**: Gli attacchi CSRF inducono un utente autenticato a inviare una richiesta indesiderata al server. Se l'applicazione non utilizza correttamente i token anti-CSRF, un attaccante potrebbe sfruttare l'utente autenticato per eseguire azioni

indesiderate.

- **Iniezione SQL (SQL Injection)**: Simile a GET, POST può anche essere vulnerabile alle iniezioni SQL se i dati inviati nel corpo della richiesta non vengono sanitizzati correttamente.
- **Iniezione di Comandi (Command Injection)**: Se i dati POST sono usati per costruire comandi shell senza convalida e sanitizzazione adeguata, un attaccante può iniettare comandi dannosi.
- **Cross-Site Scripting (XSS)**: Se i dati POST non vengono validati o filtrati correttamente, è possibile che vengano iniettati script dannosi nelle risposte.

Il metodo OPTIONS viene utilizzato per scoprire quali metodi HTTP sono supportati da un server per una risorsa specifica. Sebbene OPTIONS non trasmetta dati utente, può comunque essere sfruttato per:

- **Raccolta di Informazioni (Information Disclosure)**: Gli attaccanti possono utilizzare OPTIONS per enumerare i metodi HTTP supportati dal server e scoprire potenziali punti di attacco. Se OPTIONS rivela che metodi insicuri (come TRACE) sono abilitati, un attaccante potrebbe tentare di sfruttarli.

Il metodo TRACE è usato principalmente per il debug, consentendo di vedere il percorso di una richiesta al server e riflettendo la richiesta al mittente. È raramente utilizzato in applicazioni web moderne a causa dei rischi di sicurezza. Le vulnerabilità comuni includono:

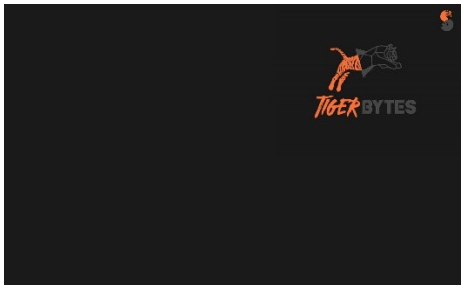
- **Cross-Site Tracing (XST)**: Un attaccante può combinare un attacco TRACE con XSS per aggirare le protezioni HttpOnly e rubare i cookie di sessione dell'utente. Questo è possibile perché la risposta del server TRACE riflette l'intera richiesta, inclusi eventuali cookie o header di autenticazione.
- **Raccolta di Informazioni (Information Disclosure)**: Se abilitato, TRACE può rivelare informazioni sensibili presenti negli header HTTP delle richieste.

Difese Generali Contro le Vulnerabilità HTTP

1. **Sanitizzazione e Validazione dell'Input**: Verifica e pulisci sempre l'input dell'utente per prevenire iniezioni SQL, XSS e altre forme di iniezione.
2. **Token Anti-CSRF**: Utilizza token anti-CSRF per proteggere le richieste POST e altre richieste che modificano lo stato dell'applicazione.
3. **Protezione dei Cookie**: Usa gli attributi HttpOnly e Secure per i

cookie per prevenire attacchi XSS e di intercettazione.

4. **Header di Sicurezza**: Utilizza header HTTP come Content-Security-Policy, X-Content-Type-Options, X-Frame-Options, e Strict-Transport-Security per proteggere le applicazioni web.
5. **Disabilita Metodi Inutilizzati**: Disabilita i metodi HTTP inutilizzati come TRACE e OPTIONS sul server per ridurre la superficie di attacco.
6. **Controllo dell'Accesso Basato sui Ruoli (RBAC)**: Implementa RBAC per garantire che solo utenti autorizzati possano accedere a determinate risorse o eseguire azioni specifiche.
7. **Logging e Monitoraggio**: Registra tutte le richieste HTTP e monitora l'attività anomala per rilevare potenziali tentativi di attacco.



Cordiali saluti,
Eugenio Levorato
Tigerbytes Company
Telefono: +39 02 87654321
Email: progettitigerbytes@gmail.com
Indirizzo: Viale dei Codici Esoterici, CAP 192168, Milano
