Juice Shop Relazione completa

2025-07-06

Contents

Contents

Introduzione

OWASP Juice Shop è un'applicazione web deliberatamente vulnerabile, progettata a fini didattici e formativi, che incorpora numerose debolezze comunemente riscontrate in contesti reali. Essa rappresenta un valido strumento per lo studio delle principali vulnerabilità elencate nella OWASP Top 10, fornendo un ambiente realistico e controllato per l'analisi della sicurezza applicativa.

Il progetto si avvale dell'immagine Docker ufficiale di OWASP Juice Shop, la quale consente di creare un ambiente isolato, facilmente replicabile e sicuro. Questa configurazione risulta particolarmente adatta per lo svolgimento di un'attività completa di Vulnerability Assessment and Penetration Testing (VAPT).

Nel presente elaborato, verranno applicate in maniera pratica le quattro fasi fondamentali del penetration testing sull'ambiente Docker dell'applicazione, illustrando le metodologie adottate, gli strumenti impiegati e i risultati ottenuti. L'obiettivo è quello di comprendere in che modo potrebbero verificarsi attacchi in scenari reali e quali strategie di prevenzione e mitigazione potrebbero essere adottate, il tutto in un contesto controllato e a scopo formativo.

Per la realizzazione del progetto è stata utilizzata la distribuzione Kali Linux, in quanto rappresenta uno standard di riferimento nel campo della sicurezza informatica. Essa include una vasta gamma di strumenti preinstallati specificamente progettati per il penetration testing, quali Burp Suite, OWASP ZAP, NMap, tra gli altri. Questo permette a Kali Linux di essere una piattaforma ideale sia per l'apprendimento che per l'applicazione pratica in ambito accademico e professionale.

Fasi del PT

Il Penetration Testing (PT) è una metodologia strutturata che consente di simulare attacchi reali su un sistema reale, con l'obiettivo di scoprire falle di sicurezza prima che possano essere sfruttate da attori malevoli. Questo processo si articola in 6 fasi principali: Pre-engagement, Information Gathering, Vulnerability Assessment, Exploitation e Post-Exploitation e Post-engagement, ciascuna con un ruolo specifico nel ciclo di valutazione della sicurezza. All'interno di questo progetto è stato eseguito un Penetration Testing che riguarda solo 4 fasi su 6 fasi totali in quanto non viene eseguito .

Information Gathering

La fase di **Information Gathering** rappresenta il primo passo fondamentale nel Penetration Testing, in cui si raccolgono tutte le informazioni utili sul target, che possono includere dati di configurazione, servizi attivi, host attivi, porte aperte. Queste informazioni sull'azienda e sull'architettura di dominio pubblico e i suoi dipendenti e altre caratteristiche rilevanti per orientare le successive attività di analisi e attacco. Nel caso dell'immagine Docker di OWASP Juice Shop, questa fase è stata svolta attraverso una serie di strumenti e tecniche, descritte di seguito in modo sequenziale.

1. Scansione dei servizi attivi con Nmap

L'obiettivo è dentificare il servizio in ascolto sul container del server locale (la porta 3000 in particolare, dove è eseguito il container Docker) e determinarne la versione confrontando le risposte del server con un database di firme. La scansione ha confermato la presenza di un server web sulla porta 3000, anche se non è stato possibile identificare con precisione la versione del servizio, probabilmente a causa dell'astrazione fornita dal container.

2. Enumerazione delle directory con ffuf

L'obiettivo è scoprire directory e file nascosti o non documentati sul server web tramite brute forcing. Usando liste di parole ad uso comune dove ogni parola della wordlist in fasi successive, usando wordlist sempre più ampie. Ogni parola viene inviata come richiesta HTTP al server per verificare l'esistenza delle risorse.

Il server rispondeva con codice 200 anche per path inesistenti, rendendo difficile distinguere le risorse valide. Per questo motivo la maggior parte delle risposte sono state filtrate, in particolare in base alla lunghezza di contenuto pari a 80117 byte, escludendo così i falsi positivi.

Questa scansione ha permesso di identificare directory effettivamente esistenti che verranno poi analizzate nella fase successiva per identificare potenziali punti d'attacco.

3. Analisi del traffico con Burp Suite

In supporto a ffuf, è stato usato Burp Suite che è stato utilizzato come proxy per intercettare e analizzare il traffico HTTP tra client e server. Ha permesso di mappare l'architettura del sito, identificare endpoint, parametri e potenziali vulnerabilità.

Questo ha permesso di mappare tutti le directory direttamente visitabili e quelle per le quali le pagine web hanno eseguito richieste specifiche. Inoltre Burp Suite permette anche di intercettare le richieste in entrata e in uscita, con la possibilità,

in modo semplificato anche tramite plugin, di modificarle per ottenere gli effetti voluti, per esempio cercando di bypassare controlli lato-client.

4. Ricognizione DNS e dominio

Tramite il comando Whois è stato fatto la raccolta di informazioni sul dominio, registrar, data di creazione e scadenza e contatti amministrativi

Tramite DNSRecon è stato possibile identificare informazioni DNS rilevanti come i name server (NS), mail server (MX), record A e AAAA (indirizzi IP).

5. Identificazione delle tecnologie con WhatWeb

Ha permesso il rilevamento di framework, librerie e configurazioni HTTP e le loro versioni associati dal quale sarà possibile effettuare scansioni per rilevare vulnerabilità non patchat e l'individuazione di potenziali configurazioni insicure, come CORS permissivi.

6. Ricerca manuale e analisi del sito web

Una parte dell'information gathering è stato effettuato in modo manuale, per idetificare punti d'attacco sfuggiti durante le scansioni automatici, sfruttando l'eventuale esperienza acquisita. Durante la navigazione manuale sono stati individuati:

- Link nascosti, come la cartella `ftp` accessibile tramite la pagina `about`.
- Enumerazione utenti tramite analisi di prodotti e pagine, identificando possibili account amministrativi.
- Stack tecnologico utilizzato (Angular, Node.js, MongoDB, ecc.) tramite l'analisi dei file JS e HTML.
- Nuove rotte scoperte analizzando i file JavaScript.
- Punti di input utente (form di login, ricerca, feedback, chatbot, ecc.).
- Credenziali hard-coded trovate nel codice sorgente.

7. Raccolta di informazioni dopo autenticazione

Successivamente è stato effettuato l'autenticazione che permette, comportandosi come un utente normale, l'accesso a nuovi punti di collegamento client-server che erano preclusi durante l'information gathering anonimo.

Durante questa fase sono state fatte, utilizzando in parallello Burp Suite:

1) Analisi delle pagine accessibili solo dopo login, come review, feedback, chatbot, complaint e carrello.
2) Osservazione delle risposte del server contenenti dati sensibili, come password hashate.
3) Presenza di un user token usato per riconoscere l'utente.

8. Analisi dei path e file scoperti

- Classificazione delle pagine trovate in base al codice di risposta HTTP (200, 301, 500).
- Esplorazione di cartelle sensibili come `ftp`, `metrics`, `api-docs`, `.well-known` e `encryptionkey`.
- Individuazione di file di configurazione e chiavi potenzialmente critiche come ad esempio la JWT key dentro encryptionkey.
- Analisi del file `robots.txt` per individuare directory nascoste.

Information gathering in dettaglio

Il capitolo rappresenta solo una riassunto dell'information gathering che è stato eseguito. Per informazioni dettagliate, riferirsi al file "**info_gathering.pdf**".

Vulnerability Assessment

Segue la fase di **Vulnerability Assessment**, in cui si identificano e classificano le vulnerabilità presenti nel sistema a partire dalle informazioni raccolte dallo step precedente. Questa fase permette di costruire una mappa delle debolezze presenti all'interno del sistema. Si possono, poi in particolare, creare delle PoC (Proof of concept) che permettono di definire, dal punto di vista teorico, eventuali strategie d'attacco che potranno poi essere effettivamente sfruttate nella fase di exploitation.

1. Accesso alla cartella FTP

Dopo aver rilevato i framework tecnologici usati dal sitor web, si è cercato di analizzare tali versioni ma senza troppo successo. Successivamente si è passato all'analisi dei path accessibili dal sito web, ed è stata scoperta la cartella `FTP` che normalmente è nascosta e protetta dagli accessi indesiderati.

È stato osservato che il server espone una cartella accessibile FTP senza richiedere autenticazione. Questo espone potenzialmente file sensibili o informazioni critiche a chiunque abbia accesso alla rete, violando i principi di confidenzialità. Al suo interno sono stati trovati file come `package.json` e `acquisition.md`, che forniscono per esempio una visione completa sull'architettura usata dal server (stack usato e le loro versioni) e informazioni confidenziali interne all'azienda.

2. Accesso ai file diversi da .md e .pdf tramite "Null Byte Injection"

Dopo aver effettuato l'accesso all'interno della path `FTP`, è stata fatta una ricognizione del path stesso che ha portato al rilevamento di file confidenziali. Si

è provato ad accedere ai vari file presenti, ma il server presentava un controllo degli accessi ai file tramite estensione.

Si è cercato di trovare un modo per bypassare questo blocco di sicurezza inducendo il server a pensare che si stia accedendo ad un file valido. È stato scoperto che inviando un input particolare costituito da una stringa che contiene `%2500`, il server legge il nome del file fino a incontrare il valore `%2500`, che viene decodificato nel valore `NULL` byte, terminando così la lettura del nome del file. Questo può permette dal punto di vista teorico l'accesso a file con estensioni diverse da `.md` e `.pdf`.

3. SQL injection tramite il login

Dopo aver constatato che il sito web permette di inviare al server degli input utente, in particolare nella pagina di `Login`, si è testata la possibilità che il server gestisca in modo sbagliato o non gestisca affatto gli input che gli arrivino. Inserendo un carattere speciale (es. ') nel campo username della pagina di login, l'applicazione ritorna un messaggio generico [object Object] e, nell'HTTP response body intercettata, compare parte della query SQL.

Questo indica che i messaggi d'errore del database non sono gestiti correttamente, rivelando dettagli interni dell'implementazione SQL e potenzialmente facilitando un attacco di SQL Injection. Inoltre, si è scoperto in questo modo che il server non esegue nessun controllo sull'input sia lato frontend sia lato backend.

4. Cross-site scripting (XSS) nella barra di ricerca

Dopo aver constatato che il sito web permette di eseguire delle ricerche sui prodotti, la barra di ricerca è stata sottoposta ad indagine per capire se esegue una qualche operazione di sanificazione e validazione degli input, per evitare l'esecuzione di script malevoli indesiderati.

È stata scoperta una vulnerabilità XSS presente nella funzione di ricerca dell'applicazione. Un utente malintenzionato può inserire codice JavaScript dannoso nel campo di ricerca, che viene poi eseguito nel browser della vittima senza adeguata sanitizzazione.

Inoltre, l'applicazione web salva il valore del campo di ricerca direttamente nell'url del sito web, permettendo ad un attaccante di inviare un link malevolo all'utente che, se accedesse al sito tramite il link, eseguirebbe involontariamente lo script malevolo.

5. Accesso ai dati del carrello degli altri utenti

Analizzando il funzionamento dell'API utilizzata per recuperare i dati del carrello dell'utente, si è osservato che essa richiede esclusivamente il basket ID

come parametro. Questo significa che, modificando manualmente il basket ID, è possibile tentare di accedere ai dati del carrello di altri utenti.

È stato scoperto che l'applicazione consente di visualizzare i carrelli di altri utenti semplicemente modificando l'ID del carrello nella richiesta. Questa esposizione diretta di un identificatore senza un adeguato controllo degli accessi permette l'accesso non autorizzato a dati sensibili.

6. Utilizzo di un algoritmo di hashing vulnerabile per le password

Dopo aver osservato che, a seguito del logout, il server restituisce informazioni relative all'account utente, si è analizzata la struttura dei dati ricevuti per determinare l'algoritmo utilizzato per l'hashing delle password.

È stato identificato l'uso dell'algoritmo MD5 per hashare le password utente senza l'aggiunta di un salt o di misure di protezione:

1) MD5 è un algoritmo obsoleto e vulnerabile ad attacchi basati su dizionario, rainbow table e brute-force.
2) Inoltre, l'assenza di un salt rende possibile precomputare gli hash e riutilizzarli per attaccare facilmente più account che condividano la stessa password.

7. Account Takeover tramite Reset Password con Domanda di Sicurezza

L'indirizzo email è stato scoperto analizzando i commenti e le recensioni pubblicate sul sito web, dove è visibile in chiaro. Questa semplice esposizione rende possibile un primo contatto con l'account, potenzialmente sfruttabile in attacchi mirati.

È stato scoperto che la funzionalità di reset password si basa su una security question legata all'utente. Tuttavia, le domande sono prevedibili, le risposte sono deboli, comuni o recuperabili da informazioni pubbliche, e non esiste un meccanismo di rate-limiting o CAPTCHA che impedisca un attacco brute-force. Un attaccante può facilmente ottenere il controllo completo dell'account bersaglio rispondendo correttamente a una domanda di sicurezza prevedibile.

8. Esposizione di Stack Trace e Messaggi di Errore Interni

Durante l'analisi delle risposte del server a richieste malformate o non gestite, è stata rilevata la presenza di messaggi di errore dettagliati contenenti stack trace e informazioni sul codice sorgente. Questi messaggi vengono mostrati direttamente nelle pagine web e rivelano dettagli tecnici sull'architettura dell'applicazione, come nomi di file, linee di codice, framework usati e struttura delle directory.

Questa esposizione non intenzionale rappresenta una Security Misconfiguration, poiché facilita un attaccante nell'identificare potenziali punti deboli e nel pianificare attacchi mirati.

9. Manipolazione del JWT

Durante l'information gathering, è stato scoperto che il server usa JWT per gestire l'identificazione dell'utente e il suo ruolo. Il token non è criptato però è encode in base64 ed ha associato a sè una firma digitale.

A seguito di analisi e modifiche, il JWT risulta essere manipolabile: in particolare modificando l'algoritmo utilizzato nell'header del token viene resa possibile anche la modifica del ruolo utente. Questo accade perchè il server legge l'algoritmo dell'header usato per firma il token e non trovando un token, semplicemente si fida del token stesso e non esegue nessun controllo sulla autenticità e sui permessi contenuti in esso.

Assenza di invalidazione del token JWT

Ulteriori analisi sul JWT toke, hanno portato a determinare che il sistema presenta una vulnerabilità critica nella gestione dei token JWT: non viene impostata una scadenza temporale (campo exp) e i token non vengono invalidati al momento del logout. Questo significa che un token, una volta emesso, rimane valido indefinitamente, anche dopo la fine della sessione utente.

Questa mancanza consente a un attaccante che riesca a ottenere un token valido (ad esempio tramite XSS o intercettazione del traffico) di continuare ad accedere al sistema senza limiti di tempo. Il rischio è elevato: si apre la porta a session hijacking, replay attack e accessi non autorizzati a dati sensibili.

In assenza di meccanismi di revoca o scadenza, il token diventa un vettore di accesso persistente, vanificando il concetto stesso di logout. È quindi fondamentale introdurre una scadenza nei token e invalidarli lato server per garantire la sicurezza delle sessioni.

10. SQL Injection nella barra di ricerca prodotti

Durante l'analisi del parametro q usato per la ricerca dei prodotti, è stata individuata una vulnerabilità di SQL Injection. Inserendo payload malevoli nel parametro, il server ha restituito errori interni (HTTP 500) o comportamenti anomali, confermando la mancata validazione dell'input lato backend.

L'utilizzo di sqlmap ha permesso di verificare la presenza di una SQL Injection blind (sia boolean-based che time-based) su database SQLite. Questa falla permette a un attaccante remoto di manipolare le query SQL ed estrarre informazioni dal database o compromettere l'app.

VA in dettaglio

Il capitolo rappresenta solo una riassunto del VA che è stato eseguito. Per informazioni dettagliate, riferirsi al file "**va.pdf**".

Exploitation

La terza fase, **Exploitation**, a partire dalle vulnerabilità catalogate nello step precedente, è possibile esegure dei tentativi di sfruttamento delle vulnerabilità individuate, andando ad eseguire le PoC definite, per ottenere un accesso non autorizzato al sistema o ai dati oppure compromettere il sistema e l'architettura stessa.

Questa fase è cruciale per verificare la reale pericolosità delle vulnerabilità rilevate e solitamente viene fatta in stretto accordo con l'azienda richiedente per evitare danni reali all'interi infrastruttura.

SQL Injection

Durante l'analisi della fase di login dell'applicazione, è emersa una vulnerabilità di SQL Injection causata dall'inserimento diretto di input utente (email, password) in query SQL non protette. L'assenza di prepared statements permette a un attaccante di manipolare il campo email per bypassare l'autenticazione, effettuando il login come admin conoscendone solo l'indirizzo email. Inserendo un payload come '–, viene ignorata la verifica della password. La vulnerabilità è facilmente riproducibile e consente accessi non autorizzati a qualsiasi account noto in particolare l'exploitation è stato fatto sull'account dell'admin.

SQL Injection – Ricerca Prodotti

È stata identificata una seconda vulnerabilità di SQL Injection sull'endpoint /rest/products/search, tramite il parametro q, che viene utilizzato direttamente all'interno di query SQL SQLite non parametrizzate. Tramite sqlmap è stato possibile effettuare un'attività di enumerazione delle tabelle nel database e successivamente estrarre contenuti sensibili dalla tabella Users. L'attacco ha utilizzato tecniche di tipo boolean-based e time-based blind. La vulnerabilità consente a un attaccante remoto, senza autenticazione, di leggere dati da qualunque tabella del db SQLite, confermando un impatto critico sulla riservatezza e integrità delle informazioni.

XSS Injection

Dalle fase di information gathering e di VA, si è scoperto che l'applicazione permette il cambio della password attraverso una richiesta GET in cui il parametro current (password attuale) è obbligatorio dal punto di vista client-side pero in

realtà è opzionale e non viene validato server-side. Questo consente a un utente autenticato di modificare la propria password senza conoscere quella attuale.

In parallelo, l'input del campo di ricerca non viene sanitizzato e consente l'iniezione di codice JavaScript via XSS. Un attaccante può quindi creato un payload contenente un `<iframe>` con uno script che modifica la password dell'utente sfruttando il token JWT salvato nel localStorage, tutto in modo invisibile all'utente bersaglio.

La sessione utente infine rimane attiva anche dopo la modifica password, rendendo lo script particolarmente pericoloso e silenzioso in quanto il token precedentemente creato rimane funzionale anche dopo che la modifica della password.

Hash password debole (MD5 senza salt)

Durante la ricognizione è stato scoperto che durante il logout, il server invia al client l'hash della password utente. Una analisi più approfondita del pacchetto inviato ha permesso di scoprire che l'algoritmo utilizzato è MD5, privo di salt, rendendo possibile il confronto diretto tra hash uguali per password uguali.

Con strumenti come John the Ripper, l'hash può essere craccato facilmente tramite un file contente un dizionario delle password più comuni sfruttando un attacco brute force. Una volta ottenuta la password in chiaro, l'attaccante può accedere liberamente all'account dell'utente.

Manipolazione del JWT (JSON Web Token)

Dopo la fase di login durante l'information gathering, è stato scoperto che, come spesso accade, il server creare un JWT per gestire le sessioni degli utenti senza la necessità per essi di inserire ad ogni accesso al sito web l'email e la password.

Durante la fase di VA si è cercato di trovare eventuali vulnerabilità che affligessero il sistema di gestione del JWT ed è stato scoperto che il sistema di autenticazione JWT presenta una configurazione errata: in alcuni casi, il server accetta token con algoritmo none, disabilitando la verifica della firma.

La fase di exploitation si concentra su questo: un attaccante può intercettare il token, deserializzarlo da base64, modificarne l'algoritmo da RSA a None e il ruolo (da "customer" ad "admin"), e inviarlo nuovamente al server. Poiché la firma non viene verificata, il token è accettato come valido, consentendo l'accesso a funzionalità riservate agli amministratori.

Questa vulnerabilità compromette completamente il modello di autorizzazione e autenticazione basato su JWT e lo rende altamente vulnerabile invece di essere usato come sistema di sicurezza per la gestione delle sessioni degli utenti.

Compromissione persistente dell'account tramite XSS e gestione inadeguata dei Token JWT

Come è stato rilevato dalla VA e come è stato poi confermato dall'exploitation, il sistema è vulnerabile a un attacco di tipo XSS persistente, che consente a un attaccante di iniettare codice JavaScript malevolo all'interno del database. Quando un utente visita una pagina contenente lo script, questo viene eseguito automaticamente dal browser e può, ad esempio, inviare il token JWT dell'utente a un server controllato dall'attaccante.

Sfruttando un'altra vulnerabilità trovata, una volta ottenuto il token, l'attaccante può usarlo per accedere al sistema come se fosse l'utente legittimo. Poiché il server non invalida mai i token, nemmeno dopo il logout o il cambio password, l'accesso rimane valido indefinitamente, permettendo un controllo persistente sull'account compromesso.

Questo tipo di vulnerabilità combina i rischi dell'XSS con quelli della cattiva gestione delle sessioni, rendendo possibile un furto silenzioso e duraturo di identità digitale. È quindi essenziale mitigare entrambi gli aspetti: prevenire l'iniezione di codice e gestire correttamente la validità dei token.

Exploitation in dettaglio

Il capitolo rappresenta solo una riassunto dell'exploitation che è stato eseguito. Per informazioni dettagliate, riferirsi al file "**exploitation.pdf**".

Post-Exploitation

Infine, la fase di **Post-Exploitation** si concentra sulle attività successive all'accesso ottenuto al fine di valutare l'impatto potenziale di un attacco riuscito e la possibilità dalle nuove informazioni raccolte eseguire attachi più profondi e mirati. Le operazioni che solitamente vengono eseguite sono:

1) Il privilege escalation
2) Il data exfiltration
3) Il persistence
4) Il detection evasion
5) La raccolta di ulteriori informazioni sensibili dall'interno del sistema (pillaging),

Data Exfiltration

Data exfiltration dei file dalla cartella FTP

È stato scoperto che la cartella FTP è accessibile pubblicamente e vulnerabile al Poison Null Byte. Questa debolezza ha permesso di scaricare automaticamente

tutti i file riservati presenti nella directory, eludendo i controlli di sicurezza.

Data exfiltration dei dati utente tramite SQL Injection

La funzione di login è risultata vulnerabile a SQL Injection, consentendo di ottenere token JWT contenenti informazioni personali degli utenti. Questi token hanno rivelato email, hash delle password, ruoli e chiavi TOTP, facilitando il furto di identità e l'accesso non autorizzato.

Estrazione estesa del database tramite SQL Injection

Dopo l'accesso iniziale ai dati della tabella Users, è stata eseguita una fase di enumerazione avanzata tramite sqlmap per valutare la portata completa della compromissione. Sono stati estratti ulteriori dati sensibili da tabelle come Cards, SecurityAnswers, Wallets, e le chiavi TOTP presenti in Users. Tali dati dimostrano la mancanza di cifratura server-side, e la presenza di informazioni critiche in chiaro, come numeri di carte di credito e segreti per l'autenticazione a due fattori. Inoltre, è stato possibile identificare utenti con privilegi elevati (admin, deluxe) a fini di impersonificazione o escalation.

Download dei dati utente dopo login

Una volta ottenuto l'accesso a un account, è stato possibile visualizzare e scaricare tutti i dati personali dell'utente direttamente dall'interfaccia web. L'applicazione consente infatti l'esportazione dei dati in formato JSON, rendendo semplice la raccolta di informazioni sensibili.

Information Gathering Internamente al Sistema (Pillaging)

Analisi del file package.json

Dopo aver scaricato i file interni, è stato analizzato il file package.json, che elenca tutte le dipendenze Node.js usate dal server. Questo ha permesso di identificare librerie obsolete o vulnerabili che potrebbero essere sfruttate per ulteriori attacchi.

Audit delle dipendenze con npm

Utilizzando il comando npm audit, è stato generato un report dettagliato che ha evidenziato 172 vulnerabilità, di cui molte ad alta o critica gravità. Queste informazioni sono utili per pianificare exploit futuri basati su dipendenze insicure.

Privilege Escalation

Escalation dei privilegi tramite manipolazione del JWT

È stato dimostrato che il token JWT può essere modificato per cambiare il ruolo dell'utente da customer ad admin. Questo è stato possibile grazie alla mancanza di firma o alla possibilità di forzare l'algoritmo a None, permettendo l'accesso a funzionalità riservate.

Persistence

Persistenza tramite JWT non invalidato

Il token JWT non viene invalidato dopo logout o cambio password, consentendo un accesso persistente. Un attaccante può quindi mantenere il controllo di un account anche dopo che l'utente ha tentato di revocare l'accesso, sfruttando token precedentemente rubati o generati.

Post-Exploitation in dettaglio

Il capitolo rappresenta solo una riassunto del post-exploitation che è stato eseguito. Per informazioni dettagliate, riferirsi al file "**post-exploitation.pdf**".

Strumenti usati

Tutte le 4 fasi sulle 6 fasi totali del Penetration Testing sono state eseguite in un ambiente Linux, in particolare, l'ambiente utilizzato è un'immagine di `Kali-Linux` preconfigurato. La scelta di questa particolare distribuzione è derivata dagli strumenti di default che Kali-linux offre per quanto riguarda la cybersecurity senza la necessità di installarli: è un ambiente pronto all'uso per il PT.

Nmap

- **Esempio di comando**:

```
nmap -sV 127.0.0.1 -p 3000
```

- **Motivo dell'utilizzo:** Identificare il servizio e la sua versione in ascolto sulla porta specificata.
- **Obiettivo della scansione:** Determinare quale applicazione (es. server web) e la sua versione è in esecuzione sulla porta 3000 del server locale (127.0.0.1). Sarebbe ovviamente possibile, eseguire una mappatura sull'IP e su tutte le porte possibili collegate all'IP stesso.

- **Spiegazione del funzionamento:** Il comando `nmap` con il flag `-sV` (Service Version detection) invia una richiesta alla porta 3000 per analizzare le risposte del servizio e confrontarle con il suo database di firme, riuscendo così a identificare il tipo di servizio e la sua versione. In questo caso, la scansione è mirata specificamente all'indirizzo localhost e alla porta 3000.

Ffuf

- **Esempio di comando**:

```
ffuf -w /usr/share/wordlists/dirb/small.txt -u
    http://127.0.0.1:3000/FUZZ -t 5
```

- **Motivo dell'utilizzo:** Eseguire un'enumerazione di directory e file (directory brute-forcing) per scoprire risorse nascoste sul server web.
- **Obiettivo della scansione:** Trovare percorsi web validi (come `/administration`, `/encryptionkey`, `/ftp`, ecc.) sul server in esecuzione all'indirizzo `http://127.0.0.1:3000` utilizzando una wordlist fornita dall'utente.
- **Spiegazione del funzionamento:** `ffuf` prende ogni parola dalla wordlist specificata (in questo caso `small.txt`) e la sostituisce al placeholder `FUZZ` nell'URL. Invia una richiesta HTTP per ogni URL generato e ne analizza la risposta. Il flag `-t 5` imposta il numero di thread a 5 per ridurre il numero di richieste contemporanee per non sovracaricare il server e cercare di ridurre eventuali blocchi anti-bot.

Burp suite

- **Motivo dell'utilizzo:** Burp viene impiegato per analizzare a fondo la struttura di un sito, identificando vulnerabilità e debolezze, grazie alla sua capacità di intercettare e manipolare il traffico tra client e server. Utilizzando Burp, è possibile simulare attacchi reali in un ambiente controllato.
- **Obiettivo della scansione:** L'obiettivo principale della scansione con Burp è quello di mappare l'intera architettura del sito web e comprendere come il traffico viene gestito. In particolare, la scansione serve a:
  - Identificare tutte le risorse, endpoint e parametri presenti nel sito.
  - Rilevare potenziali vulnerabilità come SQL injection, Cross-Site Scripting (XSS) e altre anomalie di sicurezza.
- **Spiegazione del funzionamento:** Burp agisce come un proxy tra il client (es. browser) e il server web, consentendo di intercettare e analizzare tutto il traffico HTTP/HTTPS. Esso fa:

- **Intercettazione del traffico:** tutte le comunicazioni tra il client e il server vengono catturate, permettendo agli analisti di osservare richieste e risposte in tempo reale.

- **Mappatura automatica:** durante l'intercettazione, Burp costruisce una mappa della struttura del sito web, identificando pagine, script e endpoint, oltre a raccogliere informazioni sui parametri passati.

- **Modifica in tempo reale:** si può intervenire sulle richieste HTTP per testare come il server risponde a input modificati, simulando scenari di attacco e verificando la robustezza delle difese.

whois

- **Esempio di comando**:

```
whois owasp-juice.shop
```

- **Motivo dell'utilizzo:** Raccogliere informazioni di registrazione relative al dominio principale utilizzato dall'applicazione.
- **Obiettivo della scansione:** Verificare i dati del dominio, tra cui registrar, data di creazione, scadenza, nameserver e contatti amministrativi, al fine di ottenere un primo livello di ricognizione passiva sull'infrastruttura web.
- **Spiegazione del funzionamento:** Il comando `whois` interroga i database pubblici WHOIS per ottenere informazioni sul dominio specificato. Viene restituito un set di metadati amministrativi e tecnici, che può includere contatti email, paese di registrazione, server DNS e stato del dominio.

dnsrecon

- **Esempio di comando**

```
dnsrecon -d owasp-juice.shop -t std
```

- **Motivo dell'utilizzo:** Effettuare una ricognizione DNS per raccogliere informazioni sui record associati al dominio.
- **Obiettivo della scansione:** Identificare informazioni DNS rilevanti come i name server (NS), mail server (MX), record A e AAAA (indirizzi IP), eventuali record SRV (servizi), e policy di sicurezza come DMARC e DKIM.
- **Spiegazione del funzionamento:** `dnsrecon` con l'opzione `-t std` esegue una ricognizione standard, interrogando i DNS per ottenere tutti i record

noti. È utile per avere una panoramica dell'infrastruttura e dei servizi in uso dal dominio.

whatweb

- **Esempio di comando**:

```
whatweb http://127.0.0.1:3000
```

- **Motivo dell'utilizzo:** Raccogliere informazioni sulle tecnologie e configurazioni in uso dal server web target.
- **Obiettivo della scansione:** Identificare framework, librerie JavaScript, intestazioni HTTP particolari, sistemi di gestione dei contenuti (CMS) e versioni software che potrebbero suggerire vulnerabilità note o configurazioni deboli.
- **Spiegazione del funzionamento:** `whatweb` invia una richiesta HTTP alla destinazione specificata e analizza la risposta per individuare firme associate a tecnologie web note, headers HTTP e pattern HTML.

John the ripper

- **Esempio di comando**

```
john --format=raw-md5 --wordlist=password.txt hash.txt
```

- **Motivo dell'utilizzo**: recuperare le password dell'utente a partire da hash di password ottenuti durante l'exploitation.
- **Obiettivo dell'attacco**: eseguire un attacco di tipo dictionary-based o brute-force su hash di tipo MD5, cercando di ricavare le password in chiaro attraverso un dizionario di password comuni.
- **Spiegazione del funzionamento**: John the Ripper è uno strumento avanzato di cracking password. Nell'esempio:
  - –format=raw-md5 specifica che gli hash sono codificati in formato MD5 puro.
  - –wordlist=password.txt indica la lista delle possibili password da testare.
  - hash.txt contiene gli hash bersaglio.

Il tool confronterà ogni hash con quelli generati dalle parole nel dizionario fino a trovare una corrispondenza. Con –show, vengono poi visualizzate in chiaro le credenziali recuperate.

Curl

- **Esempio di comando**

```
sh
curl -X GET http://localhost:3000/rest/user/whoami \
  -H "Authorization: Bearer <token>" \
  -H "Cookie: token=<token>" \
  -H "Content-Type: application/json" \
  -H "Accept: application/json"
```

- **Motivo dell'utilizzo**: interrogare un endpoint protetto dell'API per ottenere informazioni sull'utente autenticato, utilizzando un token JWT precedentemente ottenuto durante l'exploitation.
- **Obiettivo dell'attacco**: accedere a dati personali dell'utente (come email, ID, ruolo, ecc.) sfruttando un token valido, anche se ottenuto in modo illecito, per simulare una sessione autenticata e raccogliere informazioni sensibili.
- **Spiegazione del funzionamento**: curl consente di inviare richieste HTTP personalizzate. In questo esempio:
  - -X GET specifica che si tratta di una richiesta HTTP GET.
  - -H "Authorization: Bearer " invia il token JWT nell'header Authorization, come previsto da molte API REST.
  - -H "Cookie: token=" replica il comportamento del browser, includendo il token anche nei cookie.
  - -H "Content-Type: application/json" e -H "Accept: application/json" indicano che la richiesta e la risposta sono in formato JSON.

Questo comando è utile per verificare se un token rubato o manipolato consente ancora l'accesso a dati utente, testando così la persistenza e la portata dell'attacco.

npm audit

- **Esempio di comando**

```
npm audit --package.json > audit-report.txt
```

- **Motivo dell'utilizzo**: identificare vulnerabilità note all'interno delle dipendenze Node.js usate dall'applicazione, analizzando direttamente il file package.json ottenuto durante la fase di pillaging.
- **Obiettivo dell'attacco**: scoprire librerie obsolete o vulnerabili che potrebbero essere sfruttate per ottenere l'esecuzione di codice, accesso non autorizzato o denial of service, ampliando la superficie d'attacco.

- **Spiegazione del funzionamento**: npm audit è uno strumento integrato in Node.js che analizza le dipendenze dichiarate in package.json e package-lock.json. Nell'esempio:
  - –package.json forza l'analisi basandosi solo sulle dipendenze dichiarate, anche in assenza del file package-lock.json.
  - audit-report.txt salva l'output in un file per una revisione successiva.

Il report risultante elenca tutte le vulnerabilità trovate, suddivise per gravità, e fornisce suggerimenti su come mitigarle (es. aggiornamenti o patch). È uno strumento essenziale per valutare la sicurezza dell'ambiente Node.js compromesso.

sqlmap

- **Esempio di comando**:

```
sqlmap -u
    "http://localhost:3000/rest/products/search?q=apple"
    --level=5 --risk=3 --batch --random-agent
```

- **Motivo dell'utilizzo:** Automatizzare l'identificazione e lo sfruttamento di vulnerabilità di tipo SQL Injection al fine di esfiltrare dati dal database in modo efficiente e approfondito.
- **Obiettivo della scansione:** Accedere a dati riservati presenti nel database backend (es. email, password hash, carte di credito, chiavi TOTP, ruoli utente). Dimostrare la portata dell'iniezione SQL e l'impatto in termini di compromissione dei dati.
- **Spiegazione del funzionamento:** `sqlmap` è uno strumento avanzato che automatizza l'intero processo di individuazione ed exploit delle vulnerabilità SQLi. `-u`: indica l'URL con parametro vulnerabile (q nel motore di ricerca prodotti). `--level=5`: aumenta la profondità e l'aggressività dell'enumerazione, testando anche parametri meno evidenti o nascosti. `--risk=3`: imposta il livello di rischio massimo per consentire tecniche più invasive (es. UNION, STACKED QUERIES). `--batch`: disattiva tutte le richieste interattive, utile per l'automazione. `--random-agent`: utilizza un User-Agent casuale per evitare sistemi di difesa basati su fingerprinting del traffico. Una volta confermata la vulnerabilità, sqlmap consente di: elencare i database (`--dbs`), elencare le tabelle (`--tables`), estrarre dati specifici (`--dump`), leggere banner, utenti, password hash, ecc.

Grazie alla sua flessibilità, `sqlmap` è stato impiegato in più fasi, dalla semplice validazione della vulnerabilità fino all'esfiltrazione completa di dati altamente sensibili.

Analisi delle vulnerabilità

I dettagli delle analisi delle vulnerabilità trovate si trovano all'interno del file **va.pdf**.