

Vulnerability Assessment & Penetration Test

OWASP Juice Shop

Caroli Sofia - Passini Anna - Ricci Gabriele

Attività previste e Introduzione.....	2
Information Gathering.....	2
Descrizione delle fasi.....	2
Caratteristiche dell'applicativo.....	3
Analisi dell'immagine Docker.....	4
Struttura dell'applicativo web.....	4
Sistemi di protezione (WAF - Web Application Firewall).....	5
Studio dell'infrastruttura.....	6
Servizi dell'applicativo.....	10
Host dell'applicativo.....	10
Vulnerability Assessment.....	11
Descrizione delle fasi.....	11
Analisi delle vulnerabilità.....	11
Ricerca di dati sensibili.....	12
Vulnerability research.....	15
Creazione di Proof-Of-Concept.....	16
Conferma o confutazione delle Proof-Of-Concept.....	18
Exploitation.....	25
Descrizione delle fasi.....	25
SQL Injection & Broken Access Control.....	28
XSS Injection.....	30
Broken Access Control (domini privati).....	33
Broken Access Control & Security Misconfiguration (API e JWT).....	34
Server-Side Request Forgery (SSRF).....	36
Post-Exploitation.....	37
Descrizione delle fasi.....	37
Persistenza nell'applicativo.....	37
Lateral movement.....	39
Pillaging.....	40
Data Exfiltration.....	41
Attacchi Realistici.....	43
Normative.....	43
Conclusioni.....	44

Attività previste e Introduzione

L'obiettivo del progetto consiste nello svolgere un VAPT (Vulnerability Assessment & Penetration Testing) attraverso l'utilizzo di una macchina virtuale con sistema operativo Linux e l'installazione dell'immagine Docker [Juice Shop di OWASP](#) applicando le seguenti fasi:

- **Information Gathering;**
- **Vulnerability Assessment;**
- **Exploitation;**
- **Post-Exploitation.**

Durante tutte queste fasi abbiamo preso nota delle azioni eseguite, comandi lanciati, strumenti utilizzati, etc..., al fine di produrre un elaborato finale che sintetizzi le attività svolte, le metodologie adottate e le evidenze raccolte durante il VAPT.

OWASP Juice Shop è un applicativo web appositamente vulnerabile creato da [Björn Kimminich](#), figura di spicco nel mondo della sicurezza informatica e membro a vita di OWASP. Juice Shop è ampiamente riconosciuto in questa comunità come uno strumento di riferimento per la formazione grazie alla sua struttura modulare e alla documentazione dettagliata che lo rendono ideale per scopi didattici, permettendo di mettere in pratica tecniche di VAPT in modo sicuro ed efficace.

Information Gathering

Descrizione delle fasi

L'information gathering consiste nella ricerca e registrazione di tutte le informazioni disponibili da fonti pubbliche o tramite strumentazioni varie.

Si divide in quattro parti fondamentali:

1. **OSINT:** collezione di informazioni tramite fonti pubbliche (motori di ricerca, social media, repository pubblici...);
2. **infrastructure enumeration:** analisi della struttura IT (domini, sottodomini, indirizzi IP, rete...);
3. **service enumeration:** individuazione dei servizi attivi (porte, protocolli, versioni...);
4. **host enumeration:** fase in cui vengono esaminati i singoli host interni al perimetro d'attacco.

Abbiamo deciso di eseguire tutte le attività previste in questa fase, ma organizzeremo la relazione suddividendo i paragrafi in base alle tipologie di informazioni raccolte, piuttosto che seguire rigidamente la sequenza delle sottofasi, al fine di rendere la lettura più chiara e lineare.

Caratteristiche dell'applicativo

Abbiamo iniziato con la ricerca da fonti pubbliche: motore di ricerca, [repository di GitHub](#), [Siti ufficiali](#), [Demo dell'applicativo](#)...

Da questa prima analisi abbiamo trovato alcune informazioni:

- si tratta di un progetto **open-source**, ossia il codice sorgente dell'applicazione è pubblico e liberamente accessibile;
- è disponibile su **diverse piattaforme**, come GitHub, Docker, Heroku o installazione tramite Node.js;
- è un progetto rivolto a **principianti e studenti**, è facile da installare e include uno script che spiega le vulnerabilità (“Hacking Instructor”);
- implementa un sistema di **gamification**, infatti, presenta una bacheca per tenere traccia dei progressi sulle sfide di hacking di varia difficoltà (110 in totale);
- presenta un **supporto CTF** (Capture The Flag), ossia una competizione in cui i partecipanti devono risolvere una serie di sfide (“flag”) trovate sfruttando vulnerabilità individuate;
- è adatta al testing grazie all'**auto-riparazione**, la cache viene pulita e si ripopolata da 0 ad ogni avvio del server;
- infine, Juice Shop è **personalizzabile** in quanto può adattarsi a specifiche esigenze aziendali o dei clienti.

La pagina incoraggia la community a contribuire a codice e traduzioni e a segnalare problematiche o bug, con la possibilità di ricevere merchandising. I principali contributori e sostenitori sono:

- Björn Kimminich, l'ideatore e creatore;
- un team di volontari nella creazione e manutenzione;
- alcuni sponsor a supporto per gli eventi e la diffusione.



^ Alcuni degli sponsor che hanno contribuito a Juice Shop

Il tutto è protetto dal **diritto d'autore** dal 2014 al 2025 (MIT License). OWASP Juice Shop è strutturato in modo da offrire un primo approccio verso le principali vulnerabilità categorizzate dalla [OWASP Top 10](#) nel 2021 e, come dice il nome, simulare un vero e proprio e-commerce di succhi di frutta e merchandising relativo al sito.

Analisi dell'immagine Docker

Abbiamo iniziato l'analisi sull'applicativo tramite un controllo preliminare dell'immagine Docker, in particolare abbiamo lanciato il comando:

docker inspect juice-shop

Quest'ultimo serve per ottenere informazioni dettagliate su un'immagine Docker o un container (abbiamo denominato il nostro “juice-shop”). In particolare, abbiamo rilevato che:

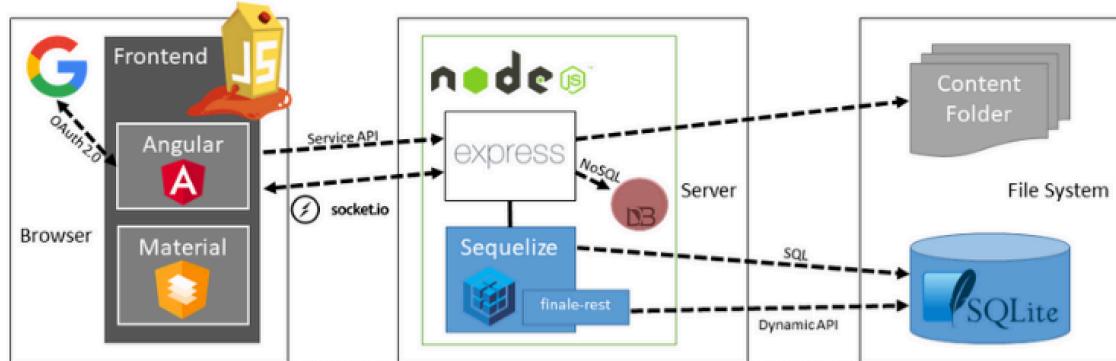
- la **porta** 3000/tcp è esposta e mappata sulla stessa porta dell'host;
- l'**indirizzo IP** assegnato al container è 172.12.0.2;
- la **versione** dell'immagine utilizzata è la 18.0.0;
- l'applicazione viene avviata tramite il **percorso** /nodejs/bin/node/juice-shop/build/app.js, da cui si deduce che è stata sviluppata utilizzando Node.js;
- è presente una **breve descrizione**: *“Probably the most modern and sophisticated insecure web application”*;
- sono forniti **link utili**, tra cui la documentazione ufficiale e il repository GitHub del progetto.

Struttura dell'applicativo web

Dalla documentazione ufficiale di OWASP Juice Shop possiamo ricavare l'istanza Demo pubblica, <https://juice-shop.herokuapp.com>: già tramite il dominio dell'URL possiamo comprendere che l'applicazione è ospitata da Heroku, un PaaS (“Platform as a Service”) che usa principalmente AWS come infrastruttura sottostante.

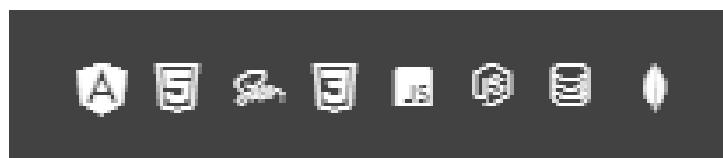
Tramite un'analisi dei file ed estensioni trovati dentro al repository di GitHub, abbiamo cercato di comprendere la struttura e le tecnologie dell'applicativo:

- **Frontend**: sviluppato con Angular e JavaScript, responsabile dell'interfaccia utente e dell'interazione client-side;
- **Backend**: basato su Express in esecuzione su un server Node.js, gestisce la logica applicativa e le API;
- **Stile**: l'aspetto grafico è curato tramite CSS3 e SASS, che permettono una gestione modulare e avanzata del design;
- **Database**: l'applicazione utilizza sia SQLite (per operazioni leggere e locali) sia MongoDB (per la gestione di dati più strutturati e persistenti);
- **Notifiche**: le notifiche relative al completamento delle sfide vengono gestite tramite WebSocket, garantendo comunicazioni in tempo reale;
- **Autenticazione**: l'accesso e la registrazione avvengono tramite OAuth 2.0 con account Google. Il sistema utilizza JSON Web Token (JWT), generati da Express, per la gestione delle sessioni di autenticazione (come configurato nel file config.yml).



^ Struttura dell'applicativo Juice Shop

Inoltre, una volta fatto l'accesso ed entrati nell'applicativo abbiamo trovato ulteriore conferma delle tecnologie utilizzate, menzionate nella sidebar:



^ Tecnologie riportate nella sidebar di Juice Shop

Infine, abbiamo utilizzato **Whatweb**, un comando di ricognizione passiva per identificare tecnologie web in uso su un sito o un'applicazione:

whatweb http://127.0.0.1:3000

Tramite l'output abbiamo individuato l'utilizzo di:

- **HTML5** per la struttura;
- **JQuery** come libreria JavaScript;
- alcuni **header HTTP non standard**;
- **header di sicurezza** che implementano misure elementari di protezione.

Sistemi di protezione (WAF - Web Application Firewall)

Per verificare la presenza di eventuali sistemi di protezione come firewall o Web Application Firewall (WAF), è stato utilizzato lo strumento **wafw00f**, eseguendo il seguente comando:

wafw00f http://juice-shop.herokuapp.com

Questo tool effettua una serie di richieste HTTP mirate per identificare la presenza di WAF noti, analizzando le risposte del server e confrontandole con firme conosciute.

L'analisi ha confermato che OWASP Juice Shop **non presenta alcun firewall o WAF attivo**, come previsto dalla sua natura: l'applicazione è infatti intenzionalmente vulnerabile. L'assenza di protezioni è una scelta deliberata, volta a consentire agli utenti di simulare attacchi reali in un ambiente controllato, favorendo l'apprendimento delle tecniche di penetration testing e delle contromisure di sicurezza.



```
~ WAFW00F : v2.2.0 ~
The Web Application Firewall Fingerprinting Toolkit

[*] Checking http://juice-shop.herokuapp.com
[+] Generic Detection results:
[-] No WAF detected by the generic detection
[~] Number of requests: 7
```

^ Output di wafw00f

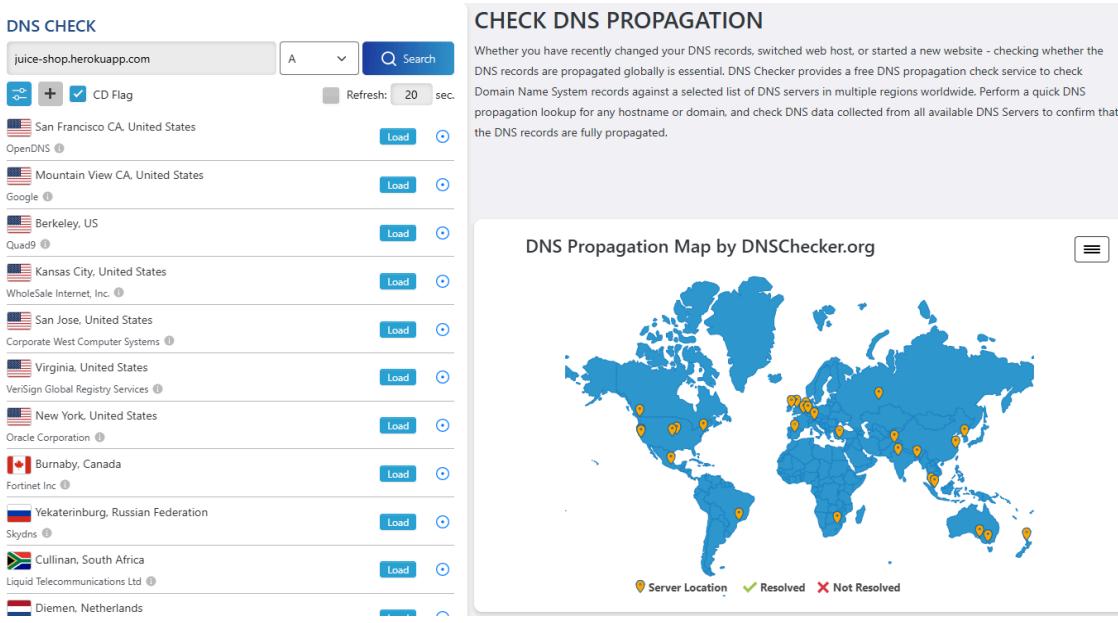
Studio dell'infrastruttura

In questa fase ci siamo concentrati sulla ricerca di IP pubblici, DNS, hostname, provider e informazioni collegate. Dato che stiamo ospitando il software su una macchina virtuale, l'unico IP esposto è quello locale; perciò sfrutteremo la Demo pubblica del software per ottenere un dominio valido e controllare relativi IPv4 e IPv6. Iniziamo lanciando il comando:

nslookup https://juice-shop.herokuapp.com

Nslookup interroga i server DNS e ottiene informazioni su nomi di dominio o indirizzi IP. Otteniamo, così, il **DNS locale** della macchina (127.0.0.53) e tre indirizzi **IP pubblici** legati a un alias verso un altro nome di dominio (54.220.192.176, 46.137.15.86, 54.73.53.134).

Abbiamo poi confermato i risultati tramite <https://dnschecker.org/>, anche questo sito interroga diversi server DNS pubblici sparsi in tutto il mondo per vedere quale indirizzo IP viene restituito per quel dominio. Tutti quegli IP puntano allo **stesso servizio**, ma sono nodi diversi della stessa infrastruttura. Abbiamo quindi capito che il dominio è risolto correttamente da server DNS in tutto il mondo, quindi l'app è **accessibile globalmente**. Essendo quindi tutti equivalenti, sceglieremo l'IP più vicino geograficamente a noi (46.137.15.86), così da avere meno latenza e scansioni più affidabili.



^ Output di dns checker

Procediamo inserendo l'IP su www.shodan.io, un motore di ricerca specializzato che consente di trovare dispositivi connessi a Internet, come server, router, webcam, sistemi industriali, IoT, etc.... Abbiamo così ricavato le seguenti informazioni:

- **Hostnames:** identificano univocamente un dispositivo o server all'interno di una rete. In questo caso, l'hostname ec2-46-137-15-86.eu-west-1.compute.amazonaws.com indica che si tratta di un'istanza EC2 pubblica con IP 46.137.15.86, situata nella regione eu-west-1 (Irlanda), e ospitata su Amazon Web Services (AWS);
- **Domains:** rappresentano nomi leggibili associati a servizi o risorse su Internet, semplificando l'accesso rispetto agli indirizzi IP. Il dominio principale rilevato è amazonaws.com;
- **Cloud Provider:** l'azienda che fornisce l'infrastruttura cloud. In questo caso, si tratta di Amazon;
- **Cloud Region:** la regione geografica del data center in cui è ospitata l'istanza, è identificata come eu-west-1, corrispondente all'Irlanda;
- **Cloud Service:** il servizio specifico utilizzato all'interno del cloud provider. In questo caso, si tratta di EC2 (Elastic Compute Cloud), che consente di eseguire macchine virtuali scalabili;
- **Country:** il paese in cui si trova fisicamente il server, ovvero Irlanda;
- **City:** la città associata alla regione cloud, ovvero Dublino;
- **Organization:** l'organizzazione responsabile dell'infrastruttura, identificata come Amazon Web Services Elastic Compute Cloud, EC2, EU;
- **ISP (Internet Service Provider):** il fornitore di connettività Internet. Poiché il server è ospitato su AWS, l'ISP coincide con il cloud provider, ovvero Amazon.com, Inc.;
- **ASN (Autonomous System Number):** è il numero identificativo del sistema autonomo che gestisce un insieme di indirizzi IP. In questo caso, l'ASN è AS16509, assegnato ad Amazon.

^ Output di Shodan (1)

Attraverso l'utilizzo della piattaforma **Shodan**, è stato possibile identificare i servizi esposti dal dominio juice-shop.herokuapp.com. In particolare, sono risultate aperte le seguenti porte:

- **Porta 80**: servizio **HTTP** (non cifrato);
- **Porta 443**: servizio **HTTPS**, cifrato tramite **TLS/SSL**.

La presenza di entrambe le porte indica che il server è configurato per accettare sia traffico **non cifrato** (HTTP) che **cifrato** (HTTPS), offrendo così compatibilità e sicurezza.

Inoltre, Shodan ha restituito informazioni dettagliate sul **certificato SSL** utilizzato:

- Il certificato è valido per *.herokuapp.com, trattandosi quindi di un **wildcard certificate**, che consente l'uso su tutti i sottodomini del dominio principale;
- **Periodo di validità**: dal 31 gennaio 2025 al 1 marzo 2026;
- **Algoritmo di firma**: SHA256 con RSA, uno standard crittografico considerato sicuro;
- **Subject Alternative Name (SAN)**: include DNS:*.herokuapp.com, confermando la validità del certificato per tutti i sottodomini associati.

Infine, le scansioni effettuate tramite Shodan confermano la presenza di istanze Juice Shop attive in diverse parti del mondo, in linea con la natura open-source e distribuita del progetto.

^ Output di Shodan (2)

Abbiamo poi utilizzato [bgp.he.net](#), uno strumento online gratuito che consente di analizzare in tempo reale il comportamento del routing BGP (Border Gateway Protocol) su Internet. Otteniamo così la lista dei **CIDR** (Classless Inter-Domain Routing), un metodo usato per rappresentare blocchi di indirizzi IP in modo compatto ed efficiente: 46.137.0.0/17 (da 46.137.0.0 a 46.137.127.255). L'IP ha inoltre 57 **host associati** (i DNS trovati prima).

[46.137.15.86 \(ec2-46-137-15-86.eu-west-1.compute.amazonaws.com\)](https://bgp.he.net/46.137.15.86)

Announced By		
Origin AS	Announcement	Description
AS16509	46.137.0.0/16	
AS16509	46.137.0.0/17	

Address has 57 hosts associated with it.

^ Output di bgp.he.net

Il sito fornisce anche un'**analisi WHOIS**, una procedura che consente di ottenere informazioni pubblicamente registrate relative a un nome di dominio o a un indirizzo IP. I dati rilevati includono:

- **NetName**: identifica il nome della rete a cui appartiene l'indirizzo IP. In questo caso, è AMAZON-EU-AWS, che indica una rete gestita da Amazon in Europa;
- **Descrizione**: fornisce informazioni aggiuntive sull'organizzazione responsabile, ovvero Amazon Web Services, EC2, EU;
- **Status**: indica lo stato del blocco IP. Il valore ASSIGNED PA (Provider Aggregatable) significa che l'intervallo IP è stato assegnato a un provider per l'uso all'interno della propria infrastruttura;
- **Maintained-by**: specifica il manutentore della rete, in questo caso Amazon Data Services, responsabile della gestione tecnica dell'intervallo IP.

```

inetnum:      46.137.0.0 - 46.137.127.255
netname:      AMAZON-EU-AWS
descr:        Amazon Web Services, Elastic Compute Cloud, EC2, EU
remarks:      The activity you have detected originates from a
              dynamic hosting environment.
              For fastest response, please submit abuse reports at
              http://aws-portal.amazon.com/gp/aws/html-forms-controller/contactus/AWSSAbuse
              For more information regarding EC2 see:
              http://ec2.amazonaws.com/
              All reports MUST include:
              * src IP
              * dest IP (your IP)
              * dest port
              * Accurate date/timestamp and timezone of activity
              * Intensity/frequency (short log extracts)
              * Your contact details (phone and email)
              Without these we will be unable to
              identify the correct owner of the IP address at that
              point in time.
country:      IE
admin-c:      ADSI2-RIPE
tech-c:       AEO1-RIPE
status:       ASSIGNED PA
mnt-by:       MNT-ADSI
mnt-domains: MNT-ADSI
created:     2010-12-02T15:38:53Z
last-modified: 2010-12-02T15:38:53Z
source:       RIPE

role:          Amazon Data Services Ireland Technical Role Account
address:      Amazon Data Services Ireland
address:      Digital Depot
address:      Thomas Street
address:      Dublin 8
address:      Ireland
mnt-by:       MNT-ADSI
tech-c:       AE2550-RIPE
nic-hdl:      ADSI2-RIPE
created:     2006-03-06T15:06:13Z
last-modified: 2023-05-16T18:13:55Z
source:       RIPE # Filtered

role:          Amazon EC2 Abuse
address:      1200 12th Avenue South
address:      Seattle
address:      WA
address:      US
mnt-by:       MNT-ADSI
admin-c:      Th510-RIPE
tech-c:       ADSI2-RIPE
nic-hdl:      AE461-RIPE
created:     2008-11-19T17:49:13Z
last-modified: 2009-12-14T11:54:25Z
source:       RIPE # Filtered

role:          Amazon EC2 Network Operations
address:      1200 12th Avenue South Seattle WA US
mnt-by:       MNT-ADSI
admin-c:      ADSI2-RIPE
tech-c:       ADSI2-RIPE
nic-hdl:      AEO1-RIPE
created:     2008-11-19T17:48:41Z
last-modified: 2019-10-02T18:36:13Z
source:       RIPE # Filtered

```

^ Analisi WHOIS di bgp.he.net

Servizi dell'applicativo

Per identificare i servizi attivi sull'host e raccogliere informazioni utili sui protocolli in uso, è stato utilizzato lo strumento **Nmap**, eseguendo il seguente comando:

Nmap -sV localhost -p 3000

Questo comando effettua una scansione della porta 3000 sull'host locale (localhost), cercando di determinare il servizio in esecuzione e la relativa versione (-sV).

La scelta della porta 3000 non è casuale: essa corrisponde alla porta su cui è mappata l'immagine Docker dell'applicazione OWASP Juice Shop. L'obiettivo è verificare se il servizio è attivo e ottenere informazioni sul software che lo gestisce.

La porta 3000 risulta aperta, confermando che il servizio è attivo. Tuttavia, **Nmap non è riuscito a identificare il servizio** in esecuzione, restituendo un risultato generico (?ppp), probabilmente a causa della mancanza di una firma nota per quel tipo di risposta.

Host dell'applicativo

L'analisi degli host serve per raccogliere informazioni dettagliate sui singoli sistemi attivi all'interno dell'infrastruttura. Abbiamo utilizzato nuovamente **Nmap**:

Nmap -sn localhost

L'output ha mostrato che sono stati scansionati 32.768 indirizzi IP, tra i quali sono stati identificati **2.428 host attivi**. La latenza molto bassa (compresa tra 0.09s e 0.35s) suggerisce che le istanze siano geograficamente vicine, tutte localizzate nella regione eu-west-1 (Irlanda) e appartenenti al dominio compute.amazonaws.com, confermando che si tratta di istanze EC2 ospitate su Amazon Web Services (AWS).

Questo dato evidenzia come l'intervallo IP sia condiviso tra più clienti AWS, una caratteristica comune nelle infrastrutture cloud pubbliche. Per motivi etici e legali, le attività di scansione sono state limitate esclusivamente all'istanza specifica che ospita l'applicazione OWASP Juice Shop.

Sebbene l'istanza analizzata risieda in un IP associato alla regione eu-west-1, è importante sottolineare che OWASP Juice Shop può essere ospitato in qualsiasi regione geografica supportata da provider cloud come AWS, Azure, Google Cloud Platform (GCP) o Heroku.

Vulnerability Assessment

Descrizione delle fasi

Il Vulnerability Assessment è un processo analitico basato su quanto è stato individuato finora, che porta a conclusioni da confermare per scartare i falsi positivi: si concentra sull'identificazione, classificazione e prioritizzazione delle vulnerabilità presenti nei sistemi informatici, fornendo una panoramica ampia ma non necessariamente approfondita delle potenziali debolezze.

L'app comprende intenzionalmente un gran numero di difetti di sicurezza (inclusi in OWASP Top 10 - 2021), ad esempio:

- Cross-Site Scripting (XSS);
- SQL Injection;
- Controllo degli Accessi Infranto (orizzontale e verticale);
- Implementazione Insecure di JWT (JSON Web Token);
- Vulnerabilità di caricamento file dovute a convalida errata dell'input.

Il Vulnerability Assessment comprende varie fasi:

1. **Vulnerability Research:** basandosi sulle informazioni raccolte, si cercano vulnerabilità, exploit e configurazioni errate già note e documentate;
2. **Causa della Vulnerabilità:** una volta identificata una potenziale vulnerabilità, si cerca di comprenderne la causa alla radice;
3. **Proof-Of-Concept (POC):** si cerca o si sviluppa un prototipo il cui scopo è testare la presenza e lo sfruttamento della vulnerabilità in modo controllato;
4. **Test della POC:** la POC viene testata per confermare in modo affidabile l'esistenza della vulnerabilità, si sottolinea l'importanza di eliminare i "falsi positivi";
5. **Iterazione:** se durante le fasi precedenti non si trova nulla di rilevante o sfruttabile, il processo ritorna alla fase di "Information Gathering" per raccogliere ulteriori dati.

Analisi delle vulnerabilità

Iniziamo con un'analisi preliminare utilizzando **Nikto**, uno strumento open-source per eseguire scansioni di sicurezza su server web. Abbiamo eseguito il comando per la scansione passando l'IP del container:

nikto -h http://172.17.0.2:3000

Lo strumento evidenzia alcune problematiche:

- **header HTTP** non comuni che non sono standard o che potrebbero rivelare informazioni non necessarie;
- **ETag** che perdono informazioni, un header usato per il caching che includono hash o ID univoci che possono rivelare dettagli sull'implementazione del server;
- il file privato “**robots.txt**”, che serve a controllare l'indicizzazione dei motori di ricerca, che rileva la presenza di un path nascosto /ftp;
- tre **directory** considerate “interessanti”: /css, /ftp/, /public/.

User-agent: *

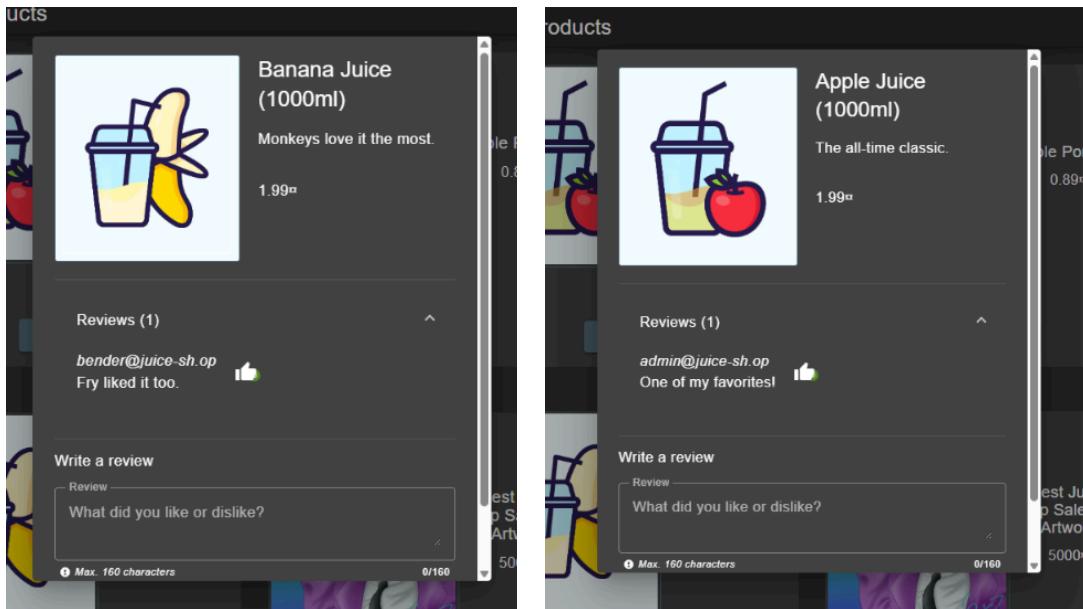
Disallow: /ftp

^ File robots.txt in Juice-Shop

Ricerca di dati sensibili

Durante l'esplorazione del sito, ci siamo concentrati sull'individuazione di dati sensibili esposti, come: e-mail, username, indizi sulle password o permessi utente.

Scorrendo le pagine abbiamo trovato la sezione delle recensioni, dove possiamo trovare una **fuga di dati**. Insieme al commento del prodotto sono riportate in chiaro le **e-mail** degli utenti; chiunque può accedere a questo dato anche senza effettuare un login.



^ Fuga di dati nelle recensioni di Juice-Shop

In particolare, abbiamo notato come, tra le varie recensioni, alcune sono lasciate da un'e-mail che potrebbe appartenere a un amministratore, **admin@juice-sh.op**, che potrebbe avere un maggior numero di permessi nell'applicativo.

Nella pagina **About Us** è presente un collegamento link (segnalato in verde) che si ricollega a una pagina esterna: si tratta del file **Legal Information**. Abbiamo notato che l'URL della nuova pagina include la cartella **ftp** (già menzionata nella ricerca di Nikto) permettendoci di navigare indietro e visitarla.

Legal Information

Lore ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accusan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consetetur adipiscig elit, sed diam nonumy nibh euismod tincidunt ut laoreet dolore magna aliquyam erat volutpat.

[^] Fuga di dati in About Us di Juice-Shop

Ci siamo successivamente spostati su Github, per analizzare il codice sorgente dell'applicativo, in particolare il file **main.js**. Qui abbiamo individuato **credenziali in chiaro**, tra cui indirizzi e-mail e password di utenti registrati. Questo rappresenta una grave vulnerabilità, in quanto consente a un attaccante di ottenere accesso diretto agli account.

```

function verifyPreLoginChallenges (req: Request) {
  challengeUtils.solveIf(challenges.passwordChallenge, () => { return req.body.email === 'admin@' + config.getString('application.domain') && req.body.password === 'admin123' })
  challengeUtils.solveIf(challenges.loginSupportChallenge, () => { return req.body.email === 'support@' + config.getString('application.domain') && req.body.password === '36avJtg0pks@5L1z2kqAVyNCE#RFSP' })
  challengeUtils.solveIf(challenges.loginRapperChallenge, () => { return req.body.email === 'mc_safearch' + config.getString('application.domain') && req.body.password === 'Mr_N0Rdles' })
  challengeUtils.solveIf(challenges.loginAnyChallenge, () => { return req.body.email === 'any@' + config.getString('application.domain') && req.body.password === 'Kif1nGh3lP' })
  challengeUtils.solveIf(challenges.dlpPasswordSprayingChallenge, () => { return req.body.email === '312934@' + config.getString('application.domain') && req.body.password === '0WBrMnuw$9VFEY$50-!fGl6t&e1B' })
  challengeUtils.solveIf(challenges.oauthUserAndPasswordChallenge, () => { return req.body.email === 'bjoren.kimminyrh@gmail.com' && req.body.password === '8W9jIamxyRlnQhJwSpbNlpaySucunvAn1' })
  challengeUtils.solveIf(challenges.exposedCredentialsChallenge, () => { return req.body.email === 'testing@' + config.getString('application.domain') && req.body.password === 'IamUsedForTesting' })
}

function verifyPostLoginChallenges (user: { data: User }) {
  challengeUtils.solveIf(challenges.loginAdminChallenge, () => { return user.data.id === users.admin.id })
  challengeUtils.solveIf(challenges.loginJIMChallenge, () => { return user.data.id === users.jim.id })
  challengeUtils.solveIf(challenges.loginHeaderChallenge, () => { return user.data.id === users.header.id })
  challengeUtils.solveIf(challenges.ghostLoginChallenge, () => { return user.data.id === users.chris.id })
  if (challengeUtils.notSolved(challenges.ephemeralAccountChallenge) && user.data.email === 'account@ntg' + config.getString('application.domain') && user.data.role === 'accounting') {
    UserModel.count({ where: { email: 'account@ntg' + config.getString('application.domain') } }).then((count: number) => {
      if (count === 0)
        challengeUtils.solveIf(challenges.ephemeralAccountChallenge)
    })
  }
}

```

^ Credenziali visibili nel file main.js

Sempre nel file main.js, è presente una mappatura dei **path accessibili** all'interno dell'applicativo. Ogni percorso presenta un attributo **canActivate**, e dai suoi valori è possibile dedurre che ogni utente possiede un ruolo specifico con permessi di accesso a specifiche aree dell'applicativo: **[ae]**, **[X]**, **[ie]**.

Il path **/administration**, in particolare, è accessibile solo agli utenti con ruolo **[ae]**.

```
, {  
    path: "administration",  
    component: cr,  
    canActivate: [ae]  
}, {  
    path: "accounting",  
    component: l1,  
    canActivate: [ie]  
}, {  
    path: "about",  
    component: Sa  
}, {  
    path: "address/select",  
    component: ys,  
    canActivate: [X]  
}, {
```

```
    }, {
      path: "address/saved",
      component: Rs,
      canActivate: [X]
    }, {
      path: "address/create",
      component: Re,
      canActivate: [X]
    }, {
      path: "address/edit/:addressId",
      component: Re,
      canActivate: [X]
    }, {
      path: "delivery-method",
      component: Yc
    }, {
      path: "deluxe-membership",
      component: Im,
      canActivate: [X]
```

```
    }, {
      path: "basket",
      component: Bo
    }, {
      path: "order-completion/:id",
      component: ac
    }, {
      path: "contact",
      component: ma
    }, {
      path: "photo-wall",
      component: Zc
    }, {
      path: "complain",
      component: Ir
    }, {
      path: "chatbot",
      component: wr
    },
  ],
  [
    {
      path: "about"
    }
  ]
]
```

```
    component: cc
  },
  {
    path: "order-summary",
    component: cc
  },
  {
    path: "order-history",
    component: xc
  },
  {
    path: "payment/:entity",
    component: dl
  },
  {
    path: "wallet",
    component: pc
  },
  {
    path: "login",
    component: xi
  },
  {
    path: "forgot-password",
    component: ei
  }
]
```

^ Path visibili nel file main.js

Cercando ancora abbiamo trovato la conferma dell'esistenza di ruoli su Juice Shop; infatti, nel file **roles.ts** sono presenti elencati tutte le possibilità:

- **customer**: probabilmente un utente base;
- **deluxe**: probabilmente un utente che ha pagato un servizio aggiuntivo;
- **accounting**: potrebbe essere un utente con permessi speciali;
- **admin**: amministratore del sito, probabilmente admin@juice-sh.op.

```
5
6      export const roles = {
7          customer: 'customer',
8          deluxe: 'deluxe',
9          accounting: 'accounting',
10         admin: 'admin'
11     }
```

^ Ruoli degli utenti in roles.ts

Infine, abbiamo trovato un secondo file, **users.yml**, che rappresenta una chiara vulnerabilità nella sicurezza. Il file da accesso, a chiunque abbia il codice sorgente, a dati come:

- **Email e password in chiaro**;
- **Ruolo utente**;
- **Domande e risposte di sicurezza**;
- **Feedback/commenti e rating**;
- **Indirizzi completi** (nome, telefono, città, stato, paese);
- **Dati di carte di credito** (nome, numero, scadenza);
- **Metadati** come walletBalance, profileImage, deletedFlag, customDomain, etc...

```
email: admin
password: 'admin123'
key: admin
role: 'admin'
securityQuestion:
  id: 2
  answer: '@x198Px00+961' |
feedback:
  comment: 'I love this shop! Best products in town! Highly recommended!'
  rating: 5
address:
  - fullName: 'Administrator'
    mobileNum: 1234567890
    zipCode: '4711'
    streetAddress: '08815 Test Street'
    city: 'Test'
    state: 'Test'
    country: 'Test'
card:
  - fullName: 'Administrator'
    cardNum: 4716190207304368
    expMonth: 2
    expYear: 2081
  - fullName: 'Administrator'
    cardNum: 4024007105648108
    expMonth: 4
    expYear: 2086
email: jim
password: 'ncc-1701'
key: jim
role: 'customer'
walletBalance: 100
securityQuestion:
  id: 1
  answer: 'Samuel' # https://en.wikipedia.org/wiki/James_T._Kirk
feedback:
  comment: 'Great shop! Awesome service!'
  rating: 4
address:
  - fullName: 'Jim'
    mobileNum: 523423432
    zipCode: '1701'
    streetAddress: 'Room 3F 121'
    city: 'Deck 5'
    state: 'USS Enterprise'
    country: 'Space'
  - fullName: 'Sam'
    mobileNum: 1000000783
    zipCode: '656783'
    streetAddress: 'Deneva Colony'
    city: 'Deneva'
    state: 'Beta Darius System'
    country: 'United Federation of Planets'
card:
  - fullName: 'Jim'
    cardNum: 5107891722778705
    expMonth: 11
email: bender
password: 'ONG&MPleaseInsertLiquori'
key: bender
role: 'customer'
securityQuestion:
  id: 10
  answer: "Stop'n'Drop" # http://futurama.wikia.com/wiki/Suicide_booth
feedback:
  comment: 'Nothing useful available here!'
  rating: 1
address:
  - fullName: 'Bender'
    mobileNum: 79765345
    zipCode: '10001'
    streetAddress: 'Robot Arms Apts 42'
    city: 'New New York'
    state: 'New New York'
    country: 'Planet Earth'
card:
  - fullName: 'Bender'
    cardNum: 4716943969046208
    expMonth: 2
    expYear: 2081
email: björn.kimminich@gmail.com
username: bkimminich
password: 'b69jLmxWlnQGhjaKpbMlpay5ucmVaml='
customDomain: true
key: björnGoogle
role: 'admin'
```

^ Dati sensibili esposti in user.yml

Vulnerability research

Una volta identificate le potenziali vulnerabilità, si cerca di comprenderne la causa alla radice: l'obiettivo è scoprire debolezze che causano le vulnerabilità e potrebbero essere sfruttate da attaccanti per compromettere la sicurezza, la riservatezza, l'integrità o la disponibilità di un sistema.

In particolare, abbiamo cercato vulnerabilità note sia appartenenti a OWASP top 10 – 2021 sia partendo delle problematiche individuate precedentemente.

1. **SQL Injection:** si tratta dell'inserimento di codice SQL malevolo per interagire con database interni. Abbiamo cercato nel sito punti con possibili vulnerabilità, partendo con l'ipotizzare injection nei vari punti di input (search, login, registrazione, review, input, chatbot, compliant). Questi punti, se non protetti, possono essere utilizzati facilmente da malintenzionati.
 - Possibile causa: mancanza di sanitizzazione e parametrizzazione dell'input.
2. **Broken Access Control:** consiste in una mal-configurazione dei controlli di accesso. Data l'analisi con Nikto, possiamo tentare di accedere alle diverse directory trovate direttamente dalla barra degli indirizzi.
 - Possibile causa: mancanza di controlli su permessi di accesso.
3. **Security Misconfiguration:** è data da configurazioni errate che rendono l'applicativo vulnerabile. Considerando che non sono seguite le impostazioni di default (es. **header HTTP non comuni**), controllare questo tipo di vulnerabilità risulta fondamentale per la sua pericolosità.
 - Possibile causa: configurazione errata e insicura.
4. **Vulnerable/Outdated Components:** uso di librerie, framework o componenti con vulnerabilità note o non aggiornati. Sempre a partire dall'analisi con Nikto, abbiamo osservato che sono presenti **ETag** che perdono informazioni e quindi risulta utile controllare se ci siano altri componenti datati.
 - Possibile causa: assenza di test e scansioni periodiche per gli aggiornamenti dei vari componenti.
5. **Identification and Authentication Failures (JWT):** consiste in problemi legati all'autenticazione o gestione della sessione. All'interno del codice abbiamo precedentemente trovato il file **config.yml**, che denota l'utilizzo di token JWT per le sessioni. Una cattiva gestione lato server dei token potrebbe compromettere l'intero sistema di autenticazione, permettendo a un attaccante di ottenere accesso non autorizzato a risorse protette.
 - Possibile causa: cattiva gestione dei token lato server.
6. **Identification and Authentication Failures (Password Dimenticata):** sempre nella stessa categoria abbiamo verificato se le password e le e-mail vengono memorizzate correttamente e permettono un accesso sicuro: l'applicazione deve anche memorizzare dati di pagamento e altri dati sensibili, quindi risulterebbe gravemente dannoso l'accesso non autorizzato. Durante l'esplorazione del sito ci siamo accorti della pagina "Password Dimenticata" che aveva una struttura non perfettamente in regola.
 - Possibile causa: meccanismi di autenticazione implementati in maniera errata.
7. **XSS injection:** sempre appartenente alla categoria injection, consiste nell'inserimento di codice (di solito JavaScript) a scopi malevoli. Anche questa è una delle più pericolose e, data la quantità di punti di input, potremmo svolgere una ricerca nei punti analoghi del SQL Injection.
 - Possibile causa: mancanza di sanitizzazione e parametrizzazione dell'input.

8. **Server Side Request Forgery (SSRF)**: si tratta di attacchi specifici appartenenti a OWASP top 10 – 2021 per la loro frequenza. Si sfrutta un server vulnerabile che viene indotto a fare richieste HTTP verso destinazioni interne o esterne non autorizzate. Abbiamo notato la possibilità di inserire URL in un form dell'applicazione web tramite la modifica del profilo: questo è un classico vettore d'ingresso per una SSRF.
- Possibile causa: errata gestione degli URL inseriti dagli utenti.

Creazione di Proof-Of-Concept

Le Proof-Of-Concept, o POC, consistono in prototipi il cui scopo è testare la presenza e la sfruttabilità della vulnerabilità in modo controllato. Abbiamo quindi proceduto a testare in vari modi le problematiche sopra evidenziate per confermarle o smentirle.

1. SQL Injection

Obiettivo: inserire del codice SQL malevolo in un punto di input.

Scenario: qualsiasi punto di input.

Payload: codice SQL che comprenda caratteri particolari (come ‘ o --).

Riproduzione:

1. selezionare un punto di input da testare;
2. inserisce il codice SQL malevolo;
3. visualizzare i risultati.

Esito atteso: l'applicazione permette l'accesso a varie righe/colonne o valori nel database.

Possibile soluzione: utilizzare query parametrizzate e validare l'input.

2. Broken Access Control

Obiettivo: accedere a dati e directory “nascoste” senza permessi.

Scenario: barra degli indirizzi.

Payload: indirizzo già presente + “/” + percorsi trovati con Nikto.

Riproduzione:

1. aggiungere un percorso trovato con Nikto;
2. visualizzare la cartella/file.

Esito atteso: l'applicazione reindirizza alla cartella scelta.

Possibile soluzione: utilizzare logging di eventi ed effettuare controlli sugli accessi.

3. Security Misconfiguration - header HTTP

Obiettivo: controllare la mancanza di sicurezza negli header HTTP.

Scenario: risposte HTTP del server.

Payload: analisi degli header di risposta.

Riproduzione:

1. ispezionare una pagina;
2. cliccare la sezione “Network/Rete”;
3. controllare gli header.

Esito atteso: non tutti gli header presenti sono standard.

Possibile soluzione: rimozione degli header non necessari e adozione solo di quelli standard.

4. Vulnerable/Outdated Components - ETag

Obiettivo: controllare la presenza di componenti obsoleti e vulnerabili.

Scenario: risposte HTTP del server.

Payload: analisi degli header di risposta.

Riproduzione:

1. ispezionare una pagina;
2. cliccare la sezione “Network/Rete”;
3. controllare e analizzare il valore degli ETag.

Esito atteso: la struttura dell'ETag è poco sicura e rivela informazioni.

Possibile soluzione: configurare il server diversamente o utilizzare hash.

5. Identification and Authentication Failures - JWT

Obiettivo: controllare il meccanismo di autenticazione basato su JWT.

Scenario: risposte HTTP del server.

Payload: intercettazione e riutilizzo del token JWT.

Riproduzione:

1. accedere al sito e autenticarsi (login o registrazione);
2. aprire la sezione “Network/Rete” degli strumenti di sviluppo;
3. individuare la richiesta che restituisce il token JWT;
4. copiare il token JWT e utilizzarlo per eseguire azioni sensibili.

Esito atteso: il token JWT è accettato senza ulteriori controlli.

Possibile soluzione: implementare meccanismi di protezione come: revoca del token, riduzione della durata o binding token/id.

6. Identification and Authentication Failures - Password Dimenticata

Obiettivo: controllare il corretto funzionamento dell'autenticazione.

Scenario: qualsiasi pagina che richieda l'inserimento di credenziali.

Payload: qualsiasi operazione che permetta di effettuare un'autenticazione non autorizzata.

Riproduzione:

1. andare nella pagina “Password Dimenticata”;
2. provare a cambiare la password corrente;
3. verificare le condizioni che lo permettono.

Esito atteso: il cambio di password non è sicuro e viene gestito in maniera errata.

Possibile soluzione: aggiunge una Multi Factor Authentication (MFA) e richiedere informazioni aggiuntive.

7. XSS injection

Obiettivo: eseguire e/o inserire codice all'interno dell'applicazione.

Scenario: qualsiasi punto di input.

Payload: script malevolo.

Riproduzione:

1. selezionare un punto di input;
2. digitare codice malevolo;
3. verificarne la presenza o visualizzarne direttamente i risultati.

Esito atteso: il codice viene inserito nella pagina.

Possibile soluzione: implementare encoding per certe tipologie di caratteri.

8. Server Side Request Forgery (SSRF)

Obiettivo: fare richieste HTTP interne ed esterne non autorizzate.

Scenario: form dell'applicazione dove inserire l'URL.

Payload: richiesta HTTP.

Riproduzione:

1. andare nella pagina del "Profilo";
2. inserire una richiesta HTTP nel form URL;
3. verificare la risposta del server.

Esito atteso: il server risponde "positivamente" alla nostra richiesta.

Possibile soluzione: effettuare monitoraggio su risposte anomale e utilizzare un allow-list.

Conferma o confutazione delle Proof-Of-Concept

Una volta stabilite le POC si procede a testarle tramite un procedimento pratico che può portare alla conferma dell'esistenza della vulnerabilità o alla sua smentita.

1. SQL Injection

Abbiamo deciso di testare la veridicità della POC controllando il codice sorgente, in particolare le query utilizzate all'interno del software. Dentro la cartella **routes/**, nel file **search.ts**, abbiamo trovato una query SELECT che seleziona i prodotti a partire da un input utente:

```
let criteria: any = req.query.q === 'undefined' ? "" : req.query.q ?? ""
criteria = (criteria.length <= 200) ? criteria : criteria.substring(0, 200)
models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE '%${criteria}%'
OR description LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER BY name`)
```

Questa query utilizza direttamente dati inseriti dagli utenti (**criteria**) senza prima sanificarli. Ciò costituisce un classico vettore d'ingresso per SQL injection: chiunque potrebbe inserire codice malevolo, come

cancellazione o modifica del database, che verrebbe messo in esecuzione senza controlli. Pertanto, confermiamo l'esistenza della vulnerabilità.

```
// vuln-code-snippet start unionSqlInjectionChallenge dbSchemaChallenge
export function searchProducts () {
  return (req: Request, res: Response, next: NextFunction) => {
    let criteria: any = req.query.q === 'undefined' ? '' : req.query.q ?? ''
    criteria = (criteria.length <= 200) ? criteria : criteria.substring(0, 200)
    models.sequelize.query(`SELECT * FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE '%${criteria}%') AND deletedAt IS NULL) ORDER BY name`)
      .then(([products]: any) => {
        const dataString = JSON.stringify(products)
        if (challengeUtils.notSolved(challenges.unionSqlInjectionChallenge)) { // vuln-code-snippet hide-start
          let solved = true
          UserModel.findAll().then(data => {
            const users = utils.queryResultToJson(data)
            if (users.data2.length) {
              for (let i = 0; i < users.data.length; i++) {
                solved = solved && utils.containsOrEscaped(dataString, users.data[i].email) && utils.contains(dataString, users.data[i].password)
                if (solved) {
                  break
                }
              }
            }
          })
        }
      })
  }
}
```

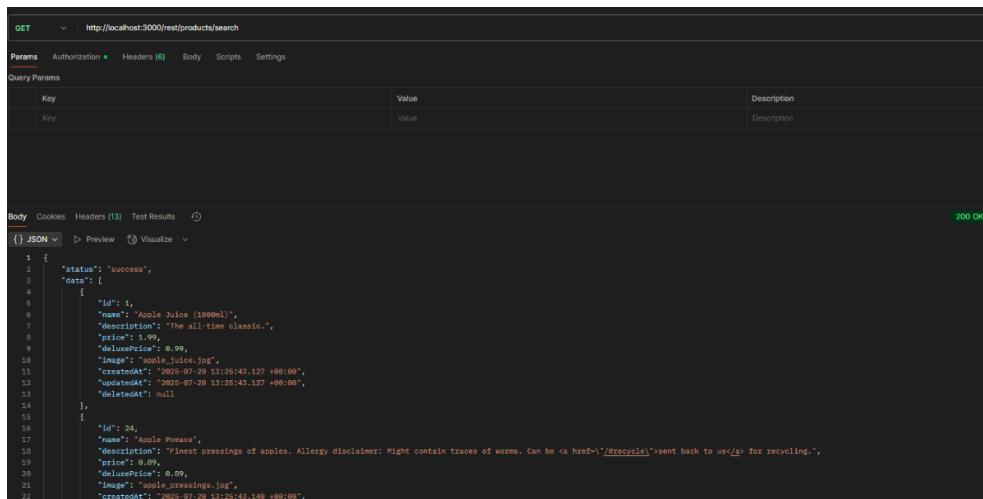
^ Query non sicura nel file search.ts

2. Broken Access Control

Secondo questa POC ci dovrebbe essere la possibilità di accedere a domini privati senza permesso. Abbiamo avviato il software e, senza eseguire l'accesso, abbiamo digitato nella barra di ricerca **http://localhost:3000/ftp**. Questa cartella era stata segnalata da Nikto, e infatti abbiamo riscontrato come sia possibile accedere a questo file di backend senza alcuna autenticazione o autorizzazione.

Inoltre, abbiamo provato a eseguire una richiesta **Postman**, senza nessun token, verso lo stesso dominio e abbiamo ottenuto un codice 200, ossia un “OK”: questo sottolinea come la risorsa sia accessibile senza controllo, e quindi vi sia una vulnerabilità in questo ambito.

Durante questa fase abbiamo verificato anche l'accesso ad alcune API interne dell'applicazione. In particolare, l'endpoint **http://localhost:3000/rest/products/search** risulta accessibile senza alcun meccanismo di autenticazione o controllo degli accessi, sia tramite browser che tramite strumenti come Postman. Tale comportamento consente a chiunque di ottenere l'elenco completo dei prodotti presenti nel database dell'applicazione, evidenziando un'ulteriore vulnerabilità di tipo Broken Access Control.



```
{ "status": "success", "data": [ { "id": 1, "name": "Apple Juice (1800ml)", "description": "The all-time classic.", "price": 1.99, "deluxePrice": 0.99, "image": "apple.juice.jpg", "createdAt": "2025-07-28 13:28:43.127 +00:00", "updatedAt": "2025-07-28 13:28:43.127 +00:00", "deletedAt": null }, { "id": 24, "name": "Apple Pomace", "description": "Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be <a href="#recycle">sent back to us</a> for recycling.", "price": 0.99, "deluxePrice": 0.99, "image": "apple.pressings.jpg", "createdAt": "2025-07-28 13:28:43.148 +00:00", "updatedAt": "2025-07-28 13:28:43.148 +00:00" } ] }
```

^ API accessibile senza autorizzazioni

3. Security Misconfiguration - header HTTP

Per controllare questa vulnerabilità abbiamo utilizzato gli strumenti di ispezione della pagina web. Nella sezione **Rete/Network** abbiamo cercato la richiesta GET per **localhost** e abbiamo controllato gli headers presenti. Alcuni header di sicurezza sono presenti (come X-Content-Type-Options, X-Frame-Options), mentre altri sono totalmente assenti (come Content-Security-Policy che proteggerebbe da XSS-injection o Strict-Transport-Security che forzerebbe l'utilizzo di HTTPS). Possiamo quindi confermare parzialmente la vulnerabilità in quanto non tutti gli headers di sicurezza sono presenti.

Nome	X	Intestazioni	Anteprima	Risposta	Iniziatore	Tempistiche	Cookie
localhost		Codice Di Stato		304 Not Modified			
cookieconsent.min...	<input checked="" type="checkbox"/>	Indirizzo Remoto		127.0.0.1:3000			
cookieconsent.min.js	<input checked="" type="checkbox"/>	Norme Sui Referrer		strict-origin-when-cross-origin			
jquery.min.js	<input checked="" type="checkbox"/>	▼ Intestazioni della risposta		Non			
runtime.js	<input checked="" type="checkbox"/>			elaborate			
polyfills.js	<input checked="" type="checkbox"/>	Accept-Ranges		bytes			
vendor.js	<input checked="" type="checkbox"/>	Access-Control-Allow-Origin		*			
main.js	<input checked="" type="checkbox"/>	Cache-Control		public, max-age=0			
cookieconsent.min...	<input checked="" type="checkbox"/>	Connection		keep-alive			
cookieconsent.min.js	<input checked="" type="checkbox"/>	Date		Sun, 20 Jul 2025 13:58:14 GMT			
jquery.min.js	<input checked="" type="checkbox"/>	Etag		W/"138f5-19828031f93"			
styles.css	<input checked="" type="checkbox"/>	Feature-Policy		payment 'self'			
en.json	<input checked="" type="checkbox"/>	Keep-Alive		timeout=5			
socket.io/?EIO=4&...	<input type="checkbox"/>	Last-Modified		Sun, 20 Jul 2025 13:25:50 GMT			
application-version	<input checked="" type="checkbox"/>	X-Content-Type-Options		nosniff			
application-config...	<input checked="" type="checkbox"/>	X-Frame-Options		SAMEORIGIN			
Challenges/?name...	<input checked="" type="checkbox"/>	X-Recruiting		/#/jobs			
languages	<input checked="" type="checkbox"/>	▼ Intestazioni delle richieste		Non			
application-version	<input checked="" type="checkbox"/>						
application-config...	<input checked="" type="checkbox"/>	▼ Intestazioni delle richieste		Non			
Challenges/?name...	<input checked="" type="checkbox"/>						

^ Ispezione degli header della pagina

4. Vulnerable/Outdated Components - ETag

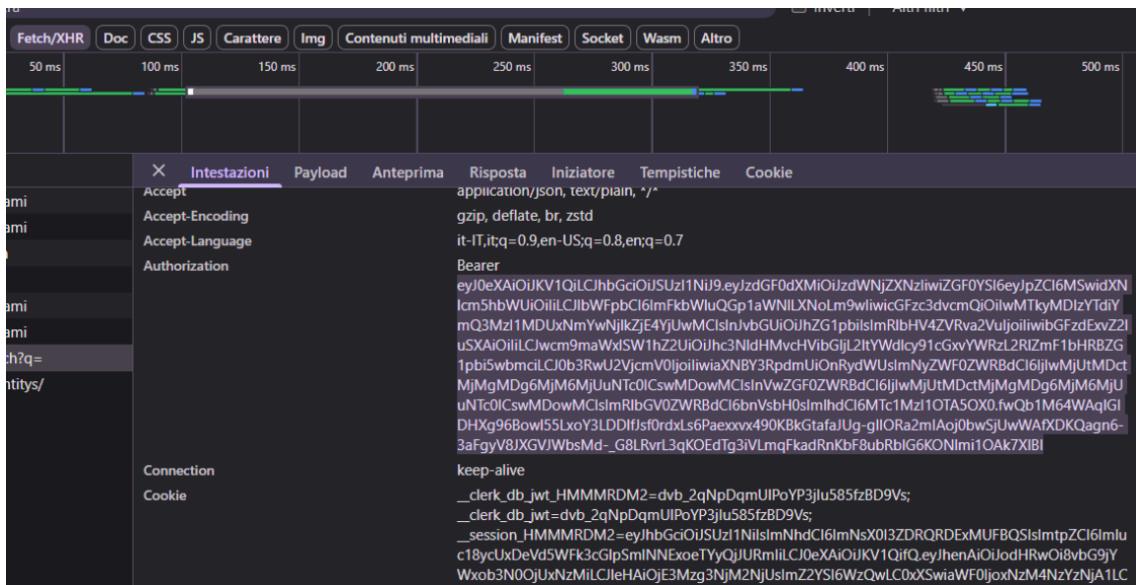
Sempre dall'ispezione precedente notiamo l'ETag: **W/"1385-19828031193"**: si tratta di un header usato per il caching che include hash o ID univoci che possono rivelare dettagli sull'implementazione del server. In particolare questo ETag è generato automaticamente da Express con la seguente struttura:

- W: sta per weak;
- 1385: la dimensione del file in byte;
- 19828031193: è il timestamp.

Si tratta di un ETag debole che rivela informazioni potenzialmente sensibili sul contenuto, come la dimensione e il timestamp. Perciò, confermiamo anche questa vulnerabilità.

5. Identification and Authentication Failures - JWT

Abbiamo testato questa PoC intercettando il **token JWT**. Questo è stato possibile eseguendo l'accesso all'applicativo e cercando nella sezione **Applicazione** dagli strumenti di ispezione del browser.



^ Token generato dopo l'autenticazione

Una volta ottenuto il token, lo abbiamo analizzato tramite <https://jwt.io>, uno strumento web che permette di ispezionare token, decodificare le varie parti e testare firme e algoritmi di codifica.

The screenshot shows the jwt.io interface. On the left, the token is displayed in its raw JSON format:

```

{
  "typ": "JWT",
  "alg": "RS256"
}

{
  "status": "success",
  "data": {
    "id": 3,
    "username": "",
    "email": "bender@juice-sh.op",
    "password": "0c36e517e3fa95aabf1bbffcc6744a4ef",
    "role": "customer",
    "deluxeToken": "",
    "lastLoginIp": "",
    "profileImage": "assets/public/images/uploads/default.svg",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2025-07-26 12:46:11.094 +00:00",
    "updatedAt": "2025-07-26 12:46:11.094 +00:00",
    "deletedAt": null
  },
  "iat": 1753535559
}

```

On the right, the 'Decoded Payload' section shows the same data structure with detailed information about each field.

^ Utilizzo di jwt.io sul token

Notiamo come il token utilizza l'algoritmo **RS256**, che prevede una firma asimmetrica:

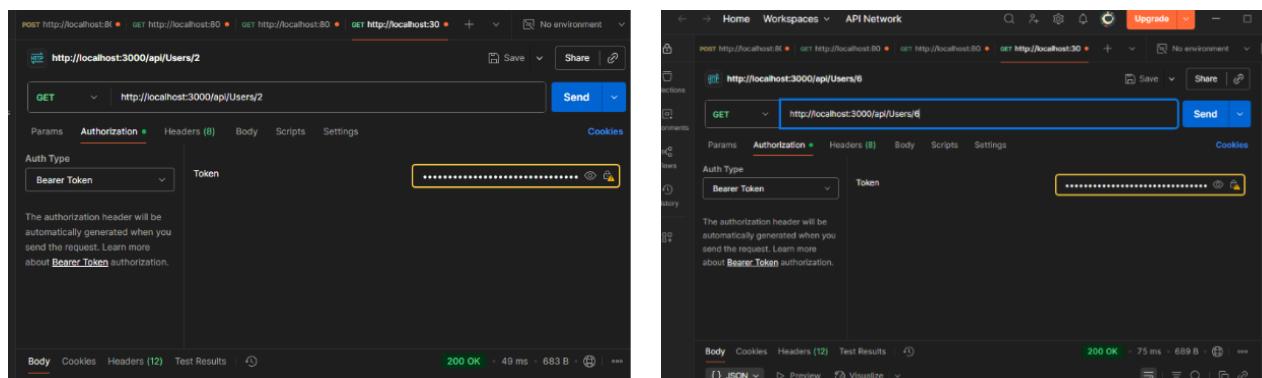
- chiave **privata** per firmare (sul server);
- chiave **pubblica** per verificare (sul client o sul server).

Questo algoritmo è considerato sicuro, a condizione che la verifica della firma sia implementata correttamente.

Tuttavia, nel payload del token, abbiamo riscontrato la presenza di **informazioni sensibili** come:

- e-mail dell'utente;
- hash della password;
- ruolo utente (role);
- stato dell'account (isActive);
- timestamp di creazione e aggiornamento.

Successivamente, abbiamo testato l'utilizzo del token per effettuare richieste GET tramite **Postman**, simulando un attacco di **session hijacking**, ovvero un furto del token di sessione. In primis, abbiamo fatto una richiesta per l'utente possessore del token, e successivamente per altri utenti. Nonostante le richieste fossero relative a utenti differenti, entrambe hanno restituito una risposta con status 200: questo comportamento suggerisce una cattiva gestione lato server dell'identità associata al token, che potrebbe permettere a un attaccante di accedere a dati sensibili di altri utenti semplicemente utilizzandone uno valido. Confermiamo anche questa POC.



^ Utilizzo del token per richieste GET verso utenti differenti

6. Identification and Authentication Failures - Password Dimenticata

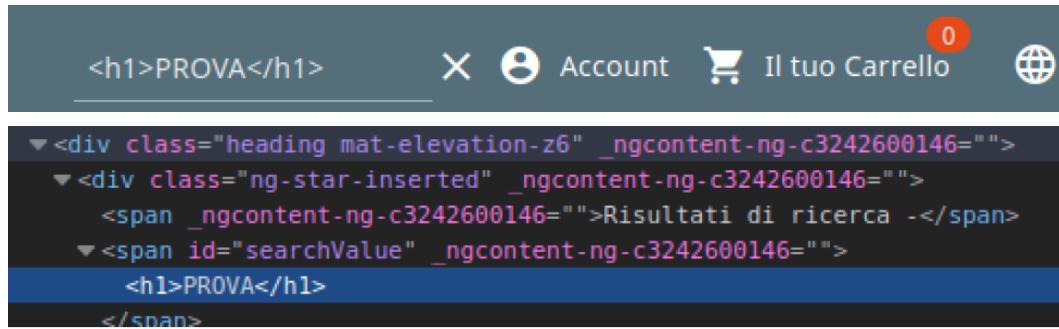
Ispezionando l'applicazione, abbiamo trovato la pagina per recuperare la password dimenticata. Per controllare se fosse possibile reperire credenziali sensibili, abbiamo provato a inserire caratteri casuali nell'input dell'indirizzo e-mail, e possiamo notare come non si ha accesso al campo successivo (**Security Question**) finché non viene inserita una e-mail salvata nel database. Questo comportamento permette di determinare quali indirizzi e-mail sono registrati nel sistema, rendendo l'input vulnerabile ad attacchi di tipo **user enumeration**: un attaccante potrebbe automatizzare l'inserimento di e-mail per identificare utenti validi.

Inoltre, conoscendo l'e-mail di un utente, possiamo provare a indovinare la sua **domanda di sicurezza** (spesso si tratta di informazioni reperibili online) e cambiare la sua password senza che ci venga richiesta quella precedente. Sottolineiamo come questo tipo di input risulti assolutamente insicuro e confermiamo la vulnerabilità.

^ Form per la password dimenticata

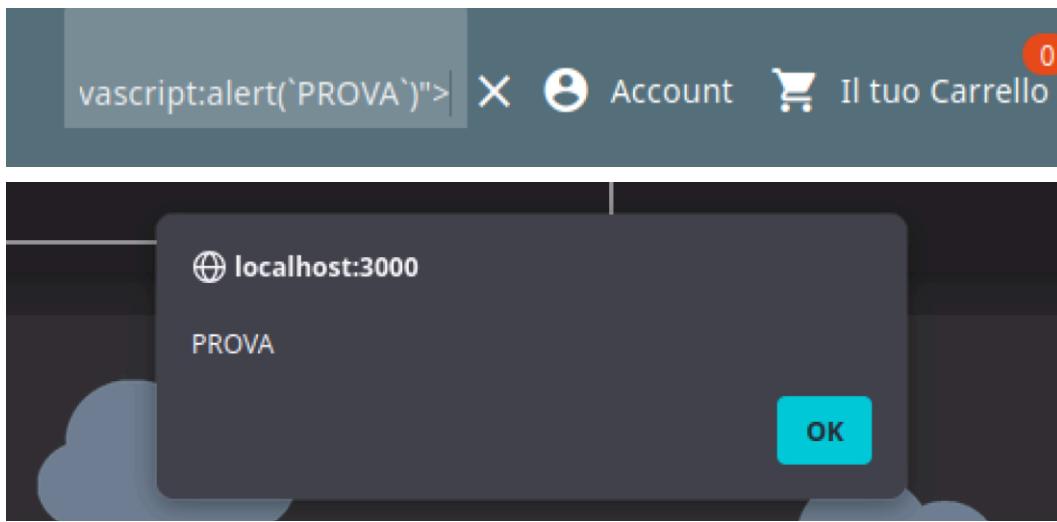
7. XSS injection

Abbiamo già notato la mancanza di header HTTP per impedire la XSS-injection, ma, per testare ulteriormente la POC, ci serviremo di un punto di input tra quelli che ci fornisce il sito. Trovato il punto adatto, ovvero la **searchbar** immediatamente visibile nella home page, procediamo a inserire un semplice codice HTML. Inviando la ricerca possiamo notare, attraverso l'ispezione browser, che il codice è stato implementato nella pagina HTML.



^ Searchbar e conseguente modifica del codice della home page

Possiamo anche provare a eseguire codice JavaScript come ulteriore prova: per vedere risultati immediati, lanciamo un **alert**, in modo che appaia un popup. Anche questa volta il codice è eseguito con successo, e abbiamo quindi un'ulteriore conferma che questa vulnerabilità è presente nell'applicativo.



^ Prova dell'inserimento di codice JavaScript nella home page

8. Server Side Request Forgery (SSRF)

Per verificare o confutare la presenza di Server-Side Request Forgery, abbiamo utilizzato il servizio <https://webhook.site>, che permette di generare un URL univoco e di ricevere e registrare richieste HTTP. Abbiamo quindi incollato l'URL nel campo **Image URL**, che avevamo identificato precedentemente nella pagina del profilo utente, e inviato il modulo. La console di Webhook restituisce un codice di stato 200: il server di Juice-Shop ha effettivamente eseguito una richiesta HTTP verso l'URL fornito. Questo comportamento è indicativo di una vulnerabilità SSRF, in quanto l'input Image URL, in un contesto reale, può essere utilizzato per inserire percorsi malevoli al fine di accedere a risorse interne alla rete, servizi cloud o per esfiltrare dati.

INBOX (1/100) Newest First		Request Details & Headers	
Search Query		Permalink API URL Open in Copy as Delete	
GET	#786cb	https://webhook.site/916b6cb1-f587-4411-b629-6010966a2d84	
137.204.150.21		Host	137.204.150.21 Whois Shodan Netify Censys VirusTotal
17/07/2025 10:35:00		Date	17/07/2025 10:35:00 (alcuni secondi fa)
		Size	0 bytes
		Time	0.000 sec
		ID	786cba2b-6163-4d1f-9a3c-1abfca93c68d
		Note	Add Note
accept-encoding: br, gzip, deflate user-agent: node sec-fetch-mode: cors accept-language: * accept: */* host: webhook.site			

^ Risposta di Webhook una volta inviato il modulo

Una volta stabilite le vulnerabilità, create e testate le POC, possiamo confermare le problematiche individuate precedentemente. Procederemo, quindi, con la fase di Exploitation.

Exploitation

Descrizione delle fasi

Ora che abbiamo un quadro specifico e confermato delle vulnerabilità, procediamo con la fase di Exploitation, che consiste nello sfruttamento delle debolezze rilevate:

- in caso di **vulnerabilità** modelliamo la POC precedentemente creata (reverse shell, exfiltration, escalation di privilegi...);
- in caso di **mis-configurationi**, le sfruttiamo direttamente (bad access rights, weak passwords).

Prioritizzazione

Per capire a quali vulnerabilità dare la precedenza, abbiamo eseguito un processo di prioritizzazione, ovvero la valutazione di tutte le vulnerabilità e debolezze trovate con varie metriche scelte ad-hoc. In particolare, abbiamo scelto di basarci su 4 parametri:

- **Successo**: probabilità che l'exploit vada a buon fine;
- **Complessità**: difficoltà nell'applicare l'exploit;
- **Danno**: probabilità di compromettere strutture o software del committente;
- **Pericolo**: rischio effettivo che quella vulnerabilità può portare al committente se sfruttata da hacker black/gray hat.

Riportiamo una tabella con spiegazione dei valori assumibili dalle metriche:

Metrica	Valore minimo	Valore massimo
Successo	1	10
Complessità	10	1
Danno	10	1
Pericolo	1	10

Una volta assegnato un valore per ogni metrica, sommandoli otterremo dei valori confrontabili che utilizzeremo per valutare le vulnerabilità più “vantaggiose” da sfruttare per l’Exploitation: un’alta probabilità di successo, una bassa complessità, una bassa probabilità di danneggiare il committente e un alto rischio per lo stesso porteranno ad avere un valore somma molto alto.

Vulnerabilità	Successo	Complessità	Danno	Pericolo	Totale
SQL injection	9	9	2	8	28
Broken Access Control (domini privati)	9	8	3	7	27
Broken Access Control (API interne)	9	8	3	7	27
Security Misconfiguration (header HTTP)	6	7	6	5	24
Vulnerable/Outdated Components (ETag)	6	7	6	5	24
Identification and Authentication Failures (JWT)	5	5	4	6	20
Identification and Authentication Failures (Password Dimenticata)	8	8	4	7	27
XSS injection	9	9	3	8	29
Server Side Request Forgery	5	5	3	7	20

Le vulnerabilità con punteggio totale più alto (che hanno maggiore rischio) sono:

- XSS Injection – Totale: 29;
- SQL Injection – Totale: 28;
- Broken Access Control (domini privati/API) – Totale: 27;
- Password Dimenticata (Auth Failure) – Totale: 27.

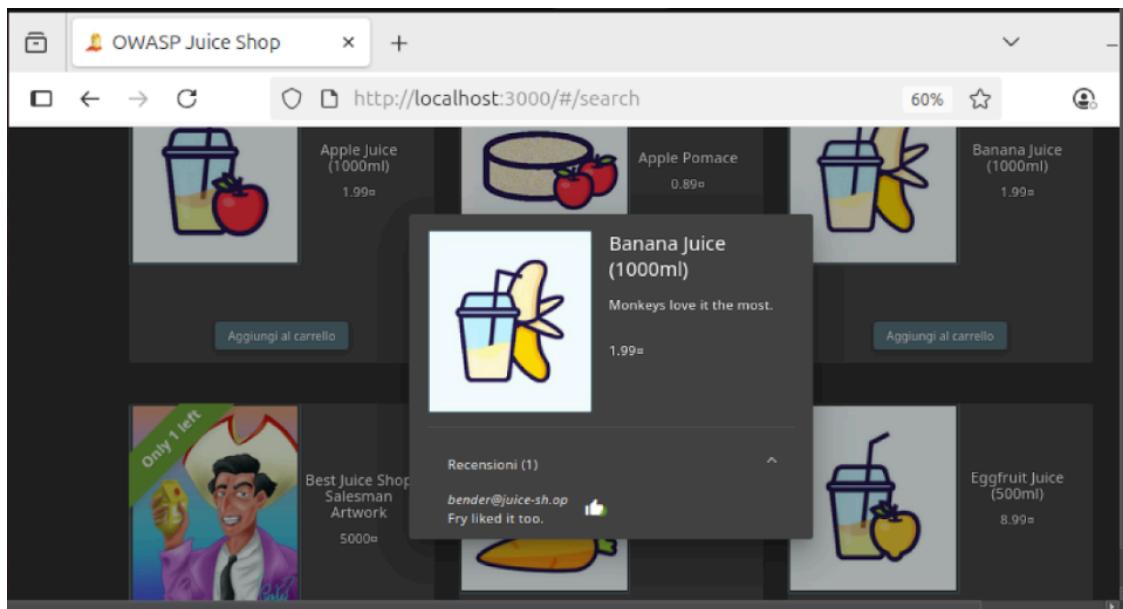
Le vulnerabilità che, secondo noi, vale la pena sfruttare sono:

1. **SQL e XSS Injection:** oltre ad essere le vulnerabilità con un totale maggiore, presenta un pericolo più alto per il committente;
2. **Broken Access Control:** sia relativi ai domini che alle API, risultano essere piuttosto pericolosi e possono causare fughe di dati e accesso a parti protette danneggiando notevolmente il committente;
3. **SSRF (Server Side Request Forgery):** nonostante risulti piuttosto complessa e con una bassa probabilità di successo, può avere forti impatti se sfruttata malevolmente.

SQL Injection & Broken Access Control

La SQL Injection (SQLi) è una delle vulnerabilità più note e pericolose nelle applicazioni web. Nella classifica OWASP Top 10 - 2021, è stata inclusa nella categoria più ampia A03:2021 – Injection, che comprende anche altri tipi di iniezione. Una SQLi si verifica quando un'applicazione consente l'inserimento di comandi SQL arbitrari all'interno delle query inviate al database, spesso a causa di input utente non correttamente validato.

Durante l'analisi dell'applicazione OWASP Juice Shop, come riportato precedentemente, abbiamo trovato una fuga di dati nella pagina dei prodotti offerti dal sito: si possono, infatti, trovare delle e-mail in chiaro dagli utenti che hanno lasciato delle recensioni.



^ Fuga di dati nella pagina dei prodotti

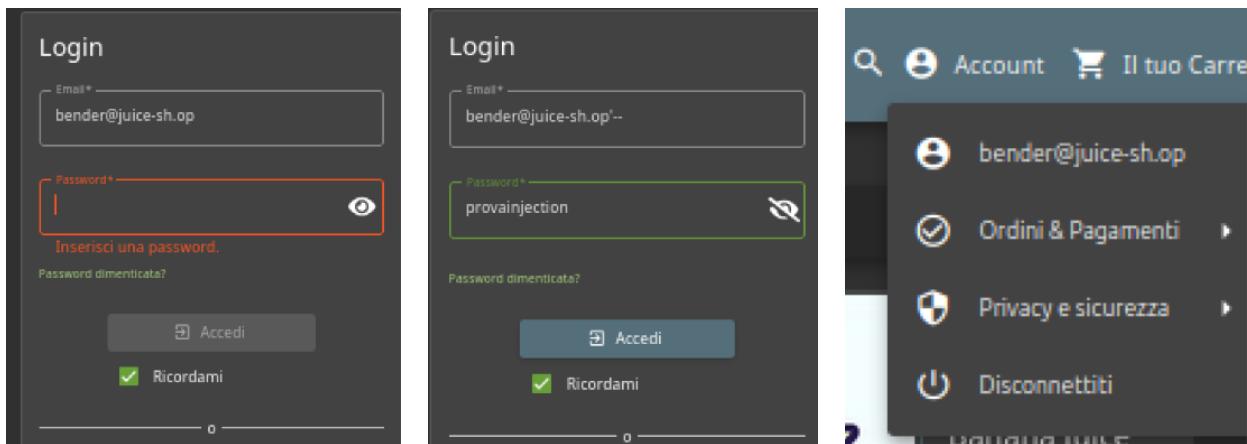
Conoscendo le e-mail degli user, ci siamo recati nella schermata **Login** e abbiamo testato il **commento SQL**: consiste nell'aggiungere ad una e-mail ' -- al fine di bypassare l'inserimento della password.

Consideriamo una classica query:

```
SELECT * FROM users WHERE username = 'user1' AND password = 'password1'
```

Inserendo ' -- dopo lo user1 andremo a commentare parte della query, eludendo il controllo della password e facendo una SELECT solo in base alla e-mail:

```
SELECT * FROM users WHERE username = 'user1' -- ' AND password = 'password1'
```

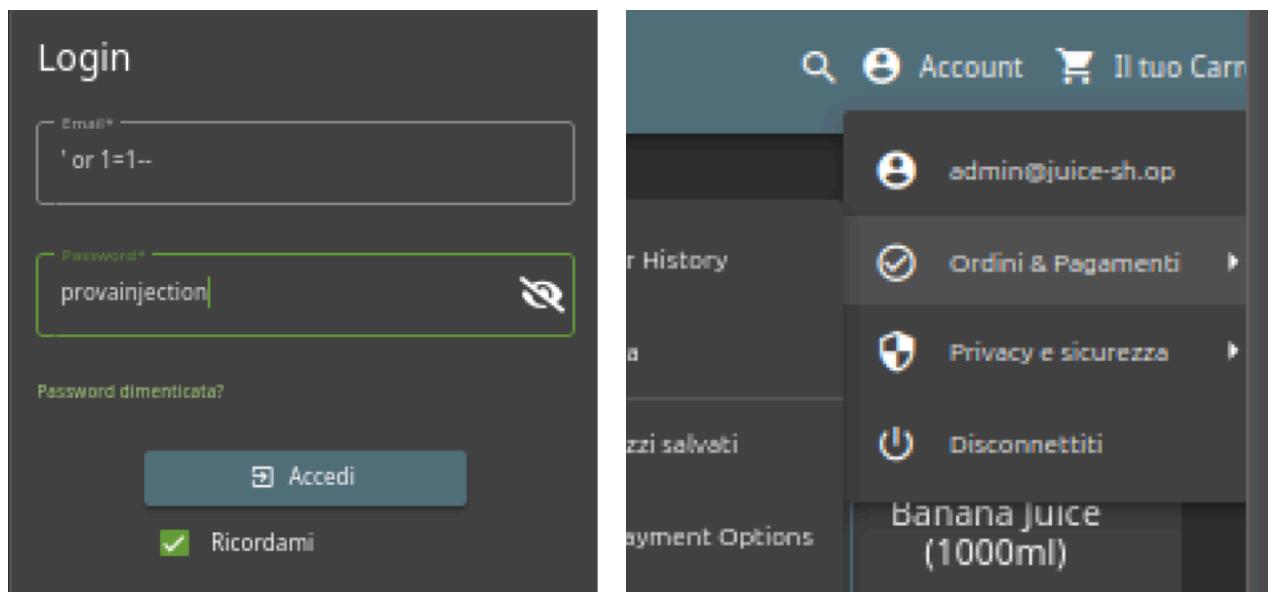


^ Logging in un altro account tramite SQLi

Un altro tipo di attacco simile si può eseguire nel campo e-mail: inserendo '**or 1=1--**' e una qualsiasi password:

SELECT * FROM users WHERE username = '' OR 1=1--' AND password

In questo caso si sfrutta **1=1**, un'espressione sempre vera che, concatenando con un OR, permette di bypassare il campo WHERE. Il login avverrà con il primo utente salvato, ovvero l'admin, una scelta fortemente pericolosa che approfondiremo più avanti.



^ SQLi per accedere all'account dell'admin senza conoscere la password

L'attacco SQL Injection rappresenta una delle minacce più insidiose per la sicurezza delle applicazioni web, in quanto può compromettere direttamente l'integrità e la riservatezza dei dati. Nel caso analizzato, l'utilizzo di tecniche come il commento SQL e l'inezione di condizioni sempre vere ha permesso di bypassare l'autenticazione e ottenere **accesso non autorizzato** (Broken Access Control), evidenziando l'importanza di una corretta gestione dell'input utente. È quindi essenziale adottare pratiche di sviluppo sicuro come:

- **query parametrizzate**: permettono di separare il codice SQL dai dati forniti dall'utente. In questo modo, anche se un utente malintenzionato inserisce del codice SQL nell'input, questo verrà trattato come semplice testo e non verrà eseguito;
- **validazione dell'input**: controllare che i dati inseriti dall'utente rispettino il formato atteso;
- **limitazione dei privilegi**: ogni componente dell'applicazione dovrebbe avere solo i privilegi strettamente necessari (ad esempio, un'applicazione che legge dati non dovrebbe avere il permesso di modificarli o cancellarli).

XSS Injection

La vulnerabilità Cross-Site Scripting (XSS) consente a un attaccante di **iniettare codice JavaScript** malevolo all'interno di pagine web visualizzate da altri utenti. Anche questa vulnerabilità è classificata nella categoria A03:2021 – Injection della OWASP Top 10, insieme alla SQL Injection, e rappresenta una minaccia significativa per la sicurezza delle applicazioni web.

All'interno di OWASP Juice Shop abbiamo individuato un punto vulnerabile all'inezione di codice JavaScript: la **barra di ricerca** nella Home page, che non è correttamente sanificata. L'attributo **[innerHTML]**, in Angular, permette di iniettare direttamente HTML nel DOM: se **searchValue** (inserito dall'utente) contiene input non sanificato, un attaccante può inserire codice malevolo.

```
<div fxLayoutAlign="center">

  <div class="table-container custom-slate">

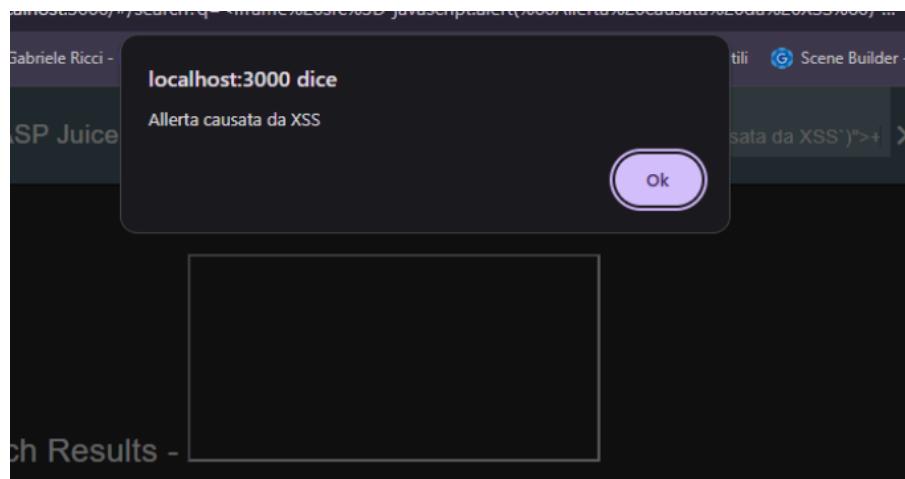
    <div class="heading mat-elevation-z6">
      <div *ngIf="searchValue">
        <span>{{"TITLE_SEARCH_RESULTS" | translate}} - </span>
        <span id="searchValue" [innerHTML]="searchValue"></span>
      </div>
      <div *ngIf="!searchValue">{{"TITLE_ALL_PRODUCTS" | translate}}</div>
      <div id="search-result-heading"></div>
    </div>
  </div>
</div>
```

^ Searchbar non correttamente sanificata

Abbiamo quindi testato vari payload per iniettare codice, partendo da:

```
<iframe src="javascript:alert('Allerta causata da XSS')">
```

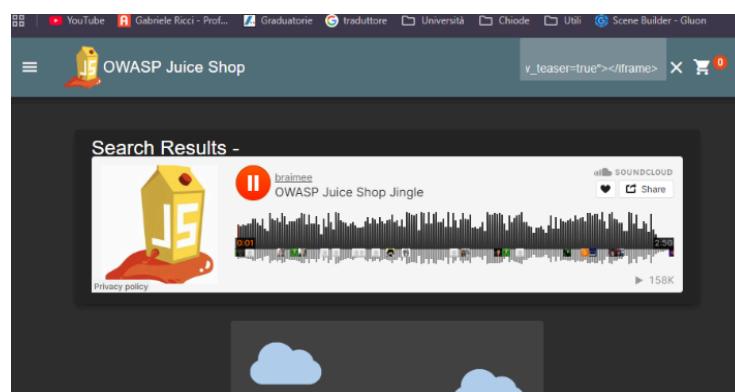
Questo payload sfrutta un **iframe**, un elemento che permette di incorporare un altro documento HTML all'interno di una pagina web: in questo caso nel campo **src** abbiamo inserito il codice JavaScript per un **alert**. Notiamo come l'input non sia filtrato correttamente, in quanto il codice va a buon fine.



^ Alert provocato dalla XSS Injection

Cercando sul web, abbiamo poi trovato una vulnerabilità nota di Juice-Shop che consente di incorporare contenuti esterni (in questo caso un **player SoundCloud**) all'interno della pagina:

```
<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>
```



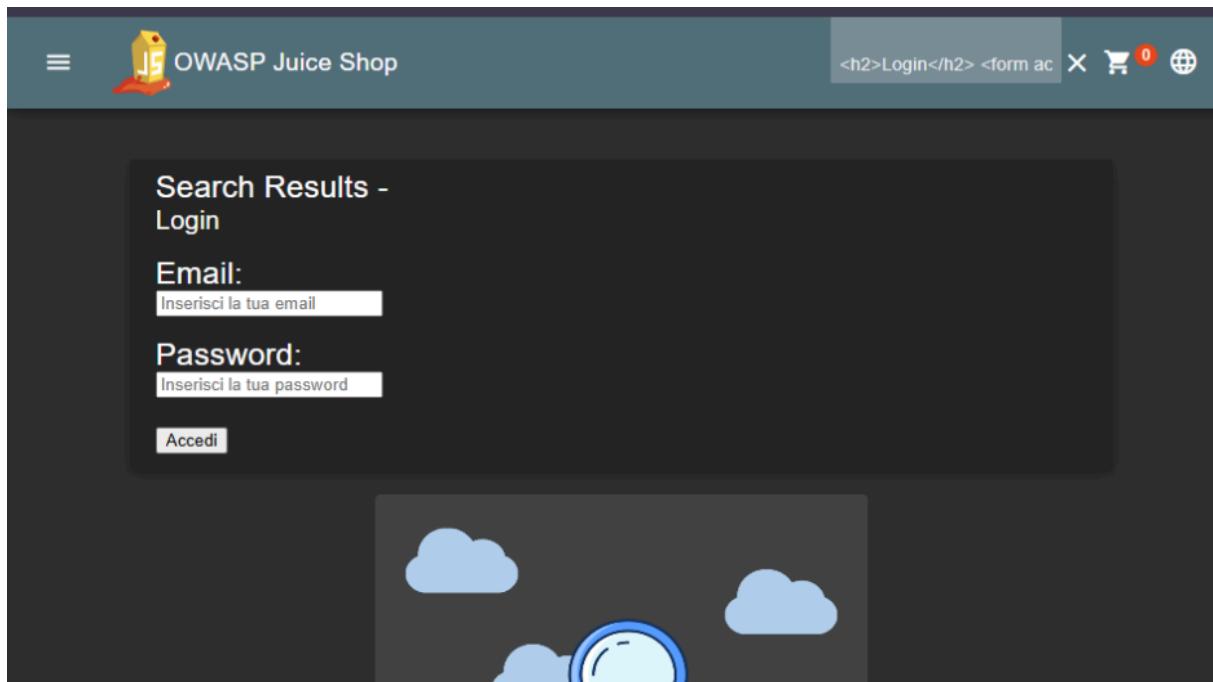
^ Inserimento di contenuto esterno nella pagina di Juice Shop

Anche se apparentemente innocui, questi esempi dimostrano come sia possibile **modificare la struttura** della pagina HTML, aprendo la strada a tecniche più avanzate di XSS. Questi attacchi possono infatti alterare la **percezione dell'interfaccia** da parte dell'utente, inducendolo a interagire con **elementi falsificati**, come form di login, pulsanti o link che sembrano legittimi ma che in realtà sono stati manipolati.

In questo modo, un attaccante può ingannare l'utente e raccogliere dati sensibili, come credenziali o informazioni personali, senza che la vittima se ne accorga, eseguendo attacchi di phishing.

Abbiamo riportato, per chiarezza, un esempio più complesso che sfrutta codice HTML per costruire un **falso form di login**:

```
<h2>Login</h2>
<form action="/login" method="POST">
    <label for="email">Email:</label><br>
    <input type="text" id="email" name="email"
placeholder="Inserisci la tua email">
    <br>
    <br>
    <label for="password">Password:</label><br>
    <input type="password" id="password" name="password"
placeholder="Inserisci la tua password"><br><br>
    <input type="submit" value="Accedi">
</form>
```



^ Inserimento di un form di login nella pagina di Juice Shop

L'impatto di un attacco XSS può essere molto grave: **furto di cookie, session hijacking, redirezione** verso siti malevoli, o esecuzione di **azioni non autorizzate** a nome dell'utente. Per mitigare questi rischi, è fondamentale quindi seguire alcune pratiche:

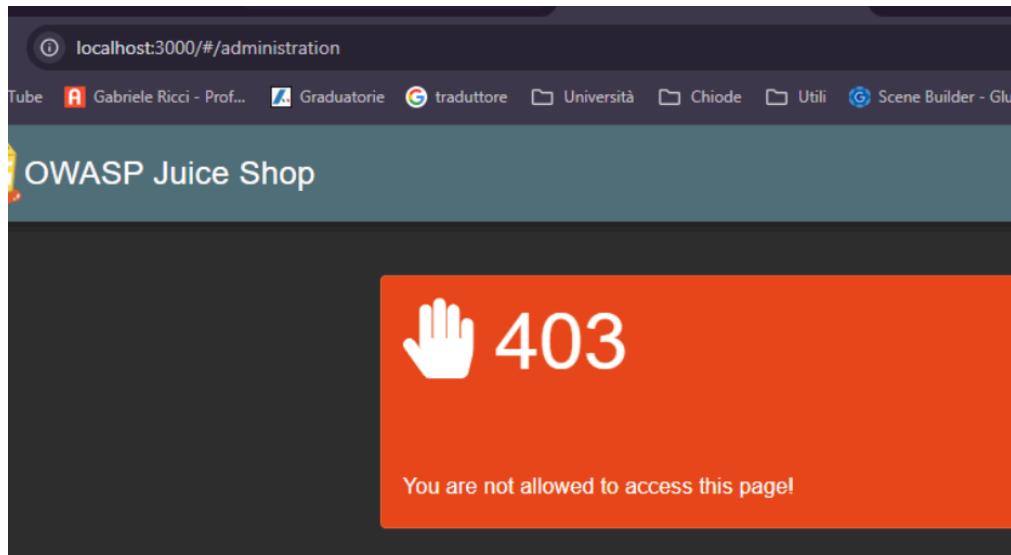
- **sanitizzare gli input:** rimuovere o neutralizzare caratteri potenzialmente pericolosi;
- **validare gli input:** assicurarsi che l'input rispetti un formato previsto;
- **librerie di escaping** per HTML, JavaScript e URL: convertire caratteri specifici in entità sicure;
- **Content Security Policy (CSP):** un'intestazione HTTP che specifica da quali fonti è consentito caricare script, immagini, fogli di stile, etc...

Broken Access Control (domini privati)

Il Broken Access Control è una delle vulnerabilità più diffuse e critiche nelle applicazioni web. Nella classifica OWASP Top 10 - 2021, è stato inserito nella categoria A01:2021 – Broken Access Control, proprio per la sua gravità e frequenza. Questa vulnerabilità si verifica quando un'applicazione non riesce a **limitare correttamente l'accesso alle risorse** in base ai privilegi, permettendo così a utenti non autorizzati di visualizzare e/o modificare **dati riservati**, accedere a **funzionalità amministrative** o **impersonare altri utenti**.

Una volta eseguito l'attacco di SQL-injection accedendo come admin, abbiamo iniziato a cercare informazioni alla quale solo questo user potesse aver accesso. Come detto in precedenza, inserire come primo utente del database l'admin non è la scelta migliore per la sicurezza, infatti ora abbiamo un accesso privilegiato al sito, e possediamo accessi a file e domini privati.

Tramite il file **main.js** abbiamo cercato domini di questo tipo, in particolare troviamo il path: <http://localhost:3000/#/administration>, che dal profilo di Bender (customer) si presenta inaccessibile.



^ Path /administration non accessibile con role: customer

Avendo l'accesso come amministratore, ripercorriamo lo stesso path e scopriamo che questa pagina contiene la **gestione delle recensioni** utenti con relative e-mail. Questa situazione mette in luce quanto possa essere pericolosa una gestione inadeguata dei controlli di accesso. Il fatto che, una volta ottenuto l'accesso come amministratore tramite una semplice SQL Injection, sia possibile raggiungere **aree riservate** e visualizzare **dati sensibili** degli utenti, dimostra una grave falla nel sistema di autorizzazione.

In particolare, l'assenza di controlli robusti lato server permette a chiunque riesca a impersonare un utente privilegiato di accedere a risorse che dovrebbero essere strettamente protette.

The screenshot shows the 'Administration' section of the VASP Juice Shop application. On the left, under 'Registered Users', there is a list of email addresses: admin@juice-sh.op, jim@juice-sh.op, bender@juice-sh.op, bjorn.kimmich@gmail.com, ciso@juice-sh.op, support@juice-sh.op, morty@juice-sh.op, mc.safesearch@juice-sh.op, J12934@juice-sh.op, and wurstbro@juice-sh.op. On the right, under 'Customer Feedback', there is a list of reviews with star ratings and short descriptions:

- 1 I love this shop! Best products in town! Highly recommended! (**in@juice-sh.op) ★★★★
- 2 Great shop Awesome service! (**@juice-sh.op) ★★★★
- 3 Nothing useful available here! (**der@juice-sh.op) ★
- 21 Please send me the juicy chatbot NFT in my wallet at /juicy-nft : "purpose betray marriage blame crunch monitor spin slide donate sport lift clutch" (**ereum@juice-sh.op) ★
- Incompetent customer support! Can't even upload photo of broken purchase! Support Team: Sorry, only order confirmation PDFs can be attached to complaints! (anonymous) ★★
- This is the store for awesome stuff of all kinds! (anonymous) ★★★★
- Never gonna buy anywhere else from now on! Thanks for the great service! (anonymous) ★★★★
- Keep up the good work! (anonymous) ★★★

^ Path administration accessibile con role: admin

Per evitare simili vulnerabilità, è fondamentale che l'applicazione non si affidi esclusivamente all'interfaccia utente per determinare i permessi, ma implementi controlli di accesso rigorosi lato backend, verificando sempre il ruolo e i privilegi dell'utente per ogni richiesta. Inoltre, è importante che i dati sensibili non siano mai esposti nel codice sorgente o accessibili tramite percorsi prevedibili.

Possono contribuire a rendere l'applicazione più sicura e resiliente contro attacchi di questo tipo:

- **corretta separazione dei ruoli:** ogni utente dovrebbe avere accesso solo alle funzionalità e ai dati coerenti con il proprio ruolo;
- **gestione delle sessioni:** token di sessione sicuri, scadenza automatica, rigenerazione del token e protezione da session fixation e session hijacking;
- **monitoraggio delle attività:** registrare e analizzare le attività degli utenti (log di accesso, modifiche ai dati, tentativi falliti, etc...).

Broken Access Control & Security Misconfiguration (API e JWT)

Un secondo esempio di Broken Access Control è stato riscontrato nell'interazione con le **API** dell'applicazione. In particolare, abbiamo sfruttato un **editor HTML** esterno per inviare richieste direttamente agli endpoint dell'API di Juice Shop, evidenziando una configurazione debole sia in termini di **access control** che di **sicurezza generale**.

Quando si utilizza un editor HTML esterno, come htmledit.squarefree.com, è possibile inviare **richieste HTTP** verso l'API dell'applicazione. In particolare, abbiamo creato una richiesta GET all'endpoint **/api/Products** e siamo riusciti a ottenere l'intero JSON dei prodotti, senza essere autenticati.

```
<!DOCTYPE html>
<html>
<head>
    <title>API Test Juice Shop</title>
</head>
<body>
    <button onclick="getProducts()">Carica Prodotti</button>
    <pre id="output"></pre>

<script>
    async function getProducts() {
        try {
            const response = await fetch('http://localhost:3000/api/Products', {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json'
                }
            });

            if (!response.ok) {
                throw new Error('Errore nella richiesta: ' + response.status);
            }

            const data = await response.json();
            document.getElementById('output').textContent = JSON.stringify(data, null, 2);
        } catch (error) {
            document.getElementById('output').textContent = 'Errore: ' + error.message;
        }
    }
</script>
</body>
</html>
```

```
{ "status": "success", "data": [ { "id": 1, "name": "Apple Juice (1000ml)", "description": "The all-time classic.", "price": 1.99, "deluxePrice": 0.99, "image": "apple_juice.jpg", "createdAt": "2025-07-17T08:27:32.658Z", "updatedAt": "2025-07-17T08:27:32.658Z", "deletedAt": null }, { "id": 2, "name": "Orange Juice (1000ml)", "description": "Made from oranges hand-picked by Uncle Dittmeyer.", "price": 2.99, "deluxePrice": 2.49, "image": "orange_juice.jpg", "createdAt": "2025-07-17T08:27:32.659Z", "updatedAt": "2025-07-17T08:27:32.659Z", "deletedAt": null } ] }
```

[^] Utilizzo di htmedit per ottenere dati dalle API

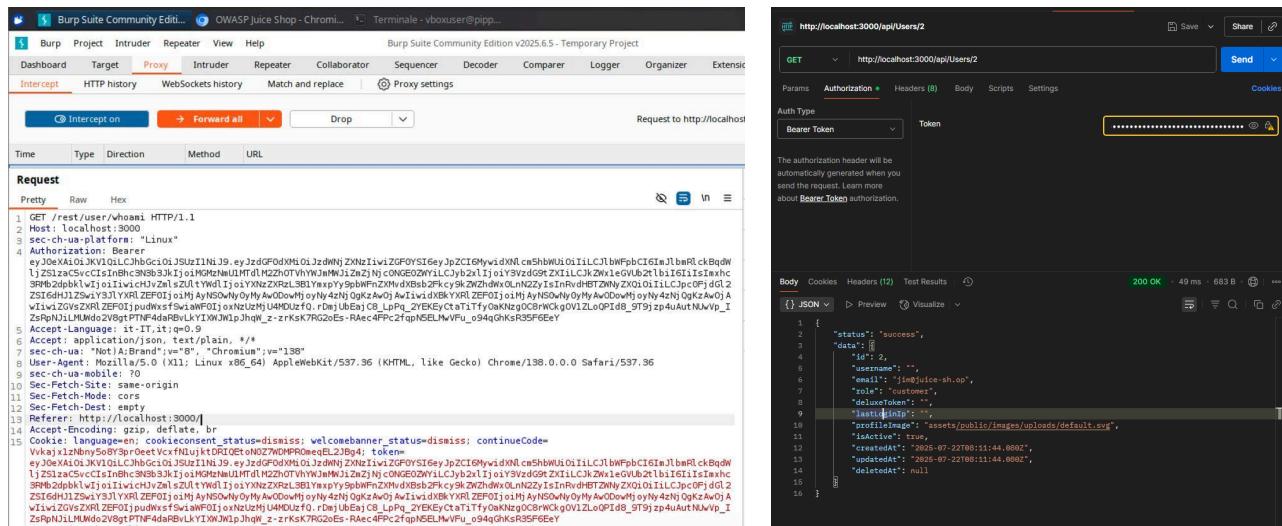
Questo è stato possibile perché l'API è accessibile **senza autenticazione**. Il server non applica correttamente le **restrizioni CORS**, un meccanismo di sicurezza implementato dai browser web per controllare le richieste tra origini diverse, permettendo a **domini esterni** di leggere le risposte.

Abbiamo tenuto in conto che questo comportamento può essere interpretato come vulnerabilità differenti

- se l'endpoint fosse intenzionalmente pubblico, si tratterebbe di una **Security Misconfiguration**;
 - se invece l'accesso dovesse essere limitato, ci troveremmo di fronte a **Broken Access Control** combinato con **Security Misconfiguration**.

La lista di prodotti in vendita poteva essere “intenzionalmente” resa pubblica, in quanto non rappresenta un’informazione sensibile, ma non si può dire lo stesso dei dati degli utenti.

Abbiamo quindi utilizzato **Postman** per eseguire una seconda richiesta GET, stavolta verso l'endpoint **/api/Users/:id**. Intercettato il token tramite **BurpSuite**, lo abbiamo inserito su Postman e abbiamo ottenuto informazioni sensibili dell'utente associato all'ID.



[^] Intercettazione del token e utilizzo dello stesso

Abbiamo poi testato con ulteriori richieste GET verso altri utenti, senza modificare il token. Le richieste sono andate tutte a buon fine dimostrando una cattiva gestione lato server dei token.

L'accesso non autenticato a **dati privati**, la **configurazione permissiva del CORS** e la possibilità di utilizzare un **token non valido** per accedere ai dati dimostrano una mancanza di controlli di sicurezza fondamentali. Questo evidenzia la necessità di:

- **autorizzazione granulari**: ogni richiesta al server dovrebbe essere verificata in base al ruolo e ai privilegi dell'utente;
- **validare il token**: per autenticità, per scadenza, per utente associato e per privilegi;
- **configurazione CORS**: per limitare le origini da cui possono essere effettuate le richieste;
- **protezione degli endpoint**: perché non devono essere facilmente individuabili e devono richiedere autenticazione e autorizzazione.

Server-Side Request Forgery (SSRF)

All'ultimo posto della classifica stilata da OWASP troviamo il Server-Side Request Forgery, categoria introdotta nell'ultima versione del 2021. SSRF è un vettore di attacco che sfrutta un'applicazione, spesso tramite HTTP (ma non solo), per eseguire una **seconda richiesta** verso risorse della rete interna ed esterna o della macchina stessa. In pratica, il server diventa un **proxy** e può essere usato per:

- accedere a risorse interne alla rete;
- interagire con servizi cloud;
- bypassare controlli di accesso o firewall;
- estrarre dati o eseguire scansioni interne.

Durante l'analisi dell'applicazione OWASP Juice Shop, abbiamo individuato un potenziale punto di ingresso per una vulnerabilità SSRF nella sezione **User Profile**. In particolare, l'applicazione consente all'utente di modificare l'immagine del profilo tramite un campo denominato **Image URL**, che accetta un indirizzo web da cui caricare l'immagine.

Abbiamo provato a inserire un URL che punta a una risorsa interna, **http://localhost:3000/ftp/coupons_2013.md.bak**, precedentemente individuata nella directory **/ftp**: l'idea è quella di forzare il server a effettuare una richiesta GET verso tale risorsa. Analizzando la risposta nella console di ispezione del browser, abbiamo osservato un codice di stato 200 OK, confermando l'avvenuto trasferimento del file.

È importante notare che, l'inserimento di virgolette ("") attorno all'URL, ha lo scopo di bypassare filtri lato server mal configurati, permettendo l'accesso a risorse interne che altrimenti verrebbero bloccate.

The screenshot shows a browser's developer tools Network tab. On the left, there is a form with an 'Image URL:' field containing 'localhost:3000/ftp/coupons_2013.md.bak'. Below it is a blue button labeled 'Link Image'. On the right, the network request details are shown:

GET http://localhost:3000/%22http://localhost:3000/ftp/coupons_2013.md.bak%22	
Stato	200 OK ?
Versione	HTTP/1.1
Trasferito	9,83 kB (dim. 80,12 kB)
Referrer Policy	strict-origin-when-cross-origin
Priorità richiesta	Low
Risoluzione DNS	Sistema

^ Richiesta del file coupons_2013.md.bak tramite SSRF

Questo esempio dimostra come un input apparentemente innocuo, come un campo per l'immagine del profilo, possa essere sfruttato per accedere a risorse interne.

Per mitigare la vulnerabilità è fondamentale implementare una serie di contromisure sia a livello applicativo che infrastrutturale. Tra le soluzioni più efficaci troviamo:

- **validazione degli URL:** assicurarsi che siano ben formati e non puntino a risorse interne o pericolose;
- **whitelist di domini:** consentono solo URL appartenenti ad una lista di domini approvati;
- bloccare l'accesso a **indirizzi IP interni o locali**;
- **proxy sicuro:** per filtrare, loggare e controllare le richieste in uscita;
- **firewall e segmentazione della rete:** per isolare il server da risorse sensibili, riducendo la superficie d'attacco;
- **monitorare le richieste in uscita:** per rilevare comportamenti anomali.

Nel caso specifico del campo **Image URL**, si potrebbe introdurre un sistema di **proxy** per il caricamento delle immagini, che verifichi e filtri preventivamente le richieste prima che vengano eseguite dal server.

Post-Exploitation

Descrizione delle fasi

Nella fase di Post-Exploitation possiamo entrare ancora di più nei panni dell'attaccante e capire come si può muovere dall'interno del sito, una volta che ha ottenuto informazioni sensibili ed è riuscito ad evadere le varie regole di sicurezza. Abbiamo 5 fasi principali:

- **Evasive Testing:** simulazione di un attacco cercando di evitare l'attivazione dei vari sistemi di difesa e rilevazione (firewall, IDS, IPS);
- **Pillaging** (information gathering dall'interno): raccolta di informazioni interne all'host;
- **Persistenza:** implementazione di un meccanismo di accesso costante, permettendo di continuare ad eseguire attività dannose;
- **Privilege Escalation:** tentativo di ottenere privilegi più elevati;
- **Data Exfiltration:** esportazione dei dati sensibili verso l'esterno.

Persistenza nell'applicativo

Juice Shop è un'applicazione web isolata e a ogni riavvio del container si possono recuperare i dati della sessione precedente: non si crea mai effettivamente un evento di disconnessione, e questo ci permette di rimanere collegati al sito senza restrizioni di tempo.

Su un applicativo non di testing (come Juice Shop), una volta ottenuto l'accesso come admin tramite SQL Injection, un attaccante potrebbe voler mantenere il controllo sull'ambiente compromesso. A questo fine abbiamo pensato a due strategie:

- **modificare le credenziali dell'admin:** semplice ma con il rischio di essere scoperto in tempi minori;
- **creazione di un nuovo utente con credenziali arbitrarie:** più complesso, ma “nascosto” e difficile da individuare.

Abbiamo quindi testato la seconda circostanza perché più complessa. Accediamo all'applicativo nella sezione registrazione, e registriamoci come nuovo utente. Intercettiamo la risposta con **Burp Suite**: l'idea è quella di modificare la risposta del server prima di fare **forward** per modificare il ruolo dell'utente appena creato, assegnandogli privilegi amministrativi (**role: admin**).

The screenshot shows the Burp Suite interface with a captured request and response. The request is a POST to '/api/users' with a JSON payload containing a user object with a role of 'customer'. The response is a 201 Created status with a JSON body containing a user object with a role of 'admin' and a new JWT token.

^ Intercettazione della richiesta di registrazione con Burp Suite

Nonostante questo tipo di attacco sia possibile, la modifica delle **response** non è attuabile senza una versione PRO di **Burp Suite**. Nel caso di Juice Shop, i ruoli utente sono gestiti tramite **token JWT**. Se il backend non verifica correttamente i privilegi associati al token, un attaccante potrebbe manipolare il payload del JWT per assegnarsi il ruolo di amministratore. Questo tipo di attacco è noto come **Privilege Escalation via JWT Tampering**.

Abbiamo poi testato la persistenza salvando localmente un token, spegnendo il container e provando a fare richieste GET riutilizzando la stessa stringa dopo il riavvio. La richiesta va a buon fine sia con il vecchio token che con quello nuovo, sintomo che la chiave segreta usata per firmare i token **non cambia al riavvio**, non esiste un meccanismo di revoca dei token attivi e non c'è un controllo su sessioni attive o logout forzato.

The screenshot shows a Postman collection with a 'users' environment. A GET request to '/api/users' is made with a 'Bearer' authorization header containing a previously saved token. The response is a 200 OK with the same JSON data as the original registration response, indicating that the token was successfully reused after a restart.

^ Riutilizzo di un token salvato in locale

Tutte queste tecniche, se effettuate su un'applicazione reale e non intenzionalmente vulnerabile come Juice Shop, rappresenterebbero un grave rischio per la sicurezza. L'assenza di controlli adeguati potrebbe esporre l'intero sistema a compromissioni, furti di dati e attacchi persistenti: è quindi fondamentale che le applicazioni implementino meccanismi di sicurezza adeguati per prevenire scenari di Post-Exploitation simili a quelli simulati in questa fase.

Lateral movement

Una volta entrato nell'applicativo, un hacker malevolo può tentare di spostarsi all'interno di una rete compromessa, da un dispositivo all'altro, alla ricerca di dati sensibili o risorse di valore. Come abbiamo scoperto precedentemente, le API non sono protette in modo appropriato e possono essere vettore di accesso per attacchi. Queste ultime sono accessibili senza autenticazione e ci hanno permesso di ottenere le informazioni di tutti gli utenti registrati nel sistema sfruttando un token qualsiasi, indipendentemente dal fatto che appartenesse a un admin o meno. Abbiamo così ricevuto la lista dei 22 utenti salvati. Sfruttando ancora le API, abbiamo poi ricavato la lista di tutti i prodotti di Juice-Shop utilizzando la stessa tecnica.

The left screenshot shows a GET request to `http://localhost:3000/api/Users`. The response body is a JSON array of 22 user objects, each with fields like id, status, username, email, role, etc. The right screenshot shows a GET request to `http://localhost:3000/rest/products/search`. The response body is a JSON array of 22 product objects, each with fields like id, name, description, price, etc.

^ Accesso alla lista dei prodotti e degli utenti da API

Siamo stati poi in grado di accedere alle informazioni del carrello di altri utenti e vederne il contenuto.

The left screenshot shows a GET request to `http://localhost:3000/rest/basket/4`. The response body is a JSON object representing a basket with a single item: Raspberry Juice (1000ml) at a price of 4.99€. The right screenshot shows the OWASP Juice Shop web interface with a message: "You successfully solved a challenge: View Basket (View another user's shopping basket.)". Below it, a modal window titled "Your Basket" shows the same basket item.

^ Accesso al carrello di altri utenti

Infine, tra gli user abbiamo individuato **accountant@juice-sh.op** con **role: accounting**: si tratta di un user con permessi speciali, ed è l'unico a poter accedere alla pagina **localhost:3000/#/accounting**. Una volta entrati nel suo account, possiamo accedere a quest'ultima pagina e scoprire che ci permette di interagire con prodotti e ordini, con azioni come:

- cambiare lo stato dei prodotti (in transito o consegnato);
- cambiare i prezzi;
- cambiare le quantità.

Track Orders			
Order ID	Price	Status	
fe01-733a10d0dc4f3e8	30.92¤	In Transit	<input checked="" type="checkbox"/>
5267-c79cb329c8434fbe	26.97¤	Delivered	<input type="checkbox"/>
5267-1390614d2d360918	8.96¤	In Transit	<input checked="" type="checkbox"/>

All Products			
Product	Price	Quantity	
Apple Juice (1000ml)	1,99	33	<input checked="" type="checkbox"/>
Apple Pomace	0,89	76	<input checked="" type="checkbox"/>
Banana Juice (1000ml)	1,99	33	<input checked="" type="checkbox"/>
Best Juice Shop Salesman Artwork	5000	1	<input checked="" type="checkbox"/>
Carrot Juice (1000ml)	2,99	65	<input checked="" type="checkbox"/>
Eggfruit Juice (500ml)	8,99	71	<input checked="" type="checkbox"/>
Fruit Press	89,99	38	<input checked="" type="checkbox"/>

^ Accesso alla pagine di accounting su Juice Shop

Le informazioni che abbiamo ottenuto possono essere fondamentali per le fasi successive.

Pillaging

Una volta ottenuta la lista di tutti gli utenti, si può esplorare il loro profilo senza conoscerne la password utilizzando ‘ -- ’ dopo lo username. Notiamo come su Juice-Shop molte informazioni sensibili siano esposte e potrebbero essere esportate per essere poi utilizzate esternamente all'applicativo (ad esempio su altri siti, dato che molte persone utilizzano sempre le stesse credenziali). Questo può portare a gravi violazioni di normative della privacy ed è per questo che è necessario utilizzare sistemi di prevenzione della perdita di dati, in particolare del Regolamento Generale sulla Protezione dei Dati (GDPR). Sono presenti dati come:

- stato, città e indirizzo;
- numero di carta con relativa scadenza;
- numero di telefono ed e-mail;
- ordini, commenti, likes...

The screenshot displays several components of the Juice Shop application:

- Request Data Export:** A modal window titled "Request Data Export" allows users to choose an export format (JSON, PDF, Excel) and includes a CAPTCHA field. It shows a JSON dump of user data.
- My saved addresses:** A list of saved addresses, including Bender (Robot Arms Apts 42, New New York, New New York, 10001).
- My Payment Options:** A list of payment cards, including one for Bender (Expiration: 2/2081).
- Add New Address:** A form for adding a new address, showing fields for Country (Planet Earth), Name (Bender), Mobile Number (797675345), ZIP Code (10001), Address (Robot Arms Apts 42), City (New New York), State (New New York), and a note about Max 160 characters.

^ Accesso a dati sensibili degli utenti su Juice Shop

Questi dati, se esfiltrati, possono essere utilizzati per attacchi di phishing, furto d'identità o frodi. Inoltre, l'esportazione non è protetta da CAPTCHA o altri meccanismi di verifica, rendendo l'operazione automatizzabile e facilmente replicabile.

Abbiamo anche sfruttato altre vulnerabilità per ottenere dati interni, previa registrazione, simulando un attacco SSRF per accedere al file **coupon_2013.md.bak**, che non dovrebbe essere visibile a utenti non autorizzati.

Data Exfiltration

Durante la fase di Post-Exploitation, abbiamo simulato diversi scenari di esfiltrazione dei dati, sfruttando le vulnerabilità precedentemente identificate. Le tecniche utilizzate sono state pensate per replicare comportamenti realistici di un attaccante che, una volta ottenuto l'accesso, mira a sottrarre informazioni sensibili.

Grazie a vulnerabilità di tipo **Broken Access Control** e **JWT mismanagement**, è stato possibile accedere ai profili utente senza autenticazione adeguata.

I dati nella fase precedente sono stati raccolti tramite chiamate API non protette, utilizzando token JWT validi, ma non associati all'utente target.

Abbiamo sfruttato la vulnerabilità SSRF nel campo “Image URL” del profilo utente per forzare il server a effettuare richieste verso risorse interne, come il file **coupon_2013.md.bak** nella directory **/ftp**: questo dimostra la possibilità di accedere a file interni e potenzialmente sensibili, anche se non esposti direttamente all'utente.

Image URL:
localhost:3000/ftp/coupons_2013.md.bak

Link Image

GET http://localhost:3000/%22http://localhost:3000/ftp/coupons_2013.md.bak%22	
Stato	200 OK ⓘ
Versione	HTTP/1.1
Trasferito	9,83 kB (dim. 80,12 kB)
Referrer Policy	strict-origin-when-cross-origin
Priorità richiesta	Low
Risoluzione DNS	Sistema

^ Richiesta del file coupons_2013.md.bak tramite SSRF

L'assenza di meccanismi di protezione interni ha permesso di enumerare gli utenti tramite la funzione “Password Dimenticata” e raccogliere quantità massive di dati tramite gli endpoint API pubblici.

```

1  {
2      "status": "success",
3      "data": [
4          {
5              "id": 1,
6              "username": "",
7              "email": "admin@juice-sh.op",
8              "role": "admin",
9              "deLuxeToken": "",
10             "lastLoginIp": "",
11             "profileImage": "assets/public/images/uploads/defaultAdmin.png",
12             "isActive": true,
13             "createdAt": "2025-07-26T12:46:11.094Z",
14             "updatedAt": "2025-07-26T12:46:11.094Z",
15             "deletedAt": null
16         },
17         {
18             "id": 2,
19             "username": "",
20             "email": "jim@juice-sh.op",
21             "role": "customer",
22             "deLuxeToken": "",
23             "lastLoginIp": "",
24             "profileImage": "assets/public/images/uploads/default.svg".

```

^ Utilizzo di API senza permessi

Tra i metodi di esfiltrazione più comuni vi è l'utilizzo di **server FTP** per salvare i dati raccolti per un utilizzo futuro: questo approccio è spesso impiegato in scenari reali per automatizzare il furto di informazioni.

Privilege Escalation

La Privilege Escalation rappresenta una delle fasi più critiche della Post-Exploitation, in cui un attaccante cerca di ottenere privilegi superiori rispetto a quelli inizialmente acquisiti. Questo consente di accedere a funzionalità riservate, manipolare dati sensibili o compromettere ulteriormente il sistema.

Nel contesto dell'applicazione Juice Shop, abbiamo simulato diversi scenari di Escalation dei privilegi:

- **accesso admin** tramite SQL Injection: sfruttando una vulnerabilità SQL, è stato possibile autenticarsi come amministratore, ottenendo così accesso completo al pannello di controllo e alle funzionalità riservate;
- **JWT tampering**: modificando il payload di un token JWT, abbiamo assegnato manualmente il ruolo admin a un utente appena registrato. Questo è stato possibile grazie alla mancata validazione lato server della firma del token;

- **ruoli speciali:** tramite tecniche di lateral movement e analisi delle API, abbiamo identificato utenti con ruoli privilegiati (es. accountant@juice-sh.op) e ottenuto accesso a funzionalità critiche come la modifica degli ordini e dei prezzi;
- **accesso a dati di altri utenti:** sfruttando ID prevedibili o token JWT riutilizzati, è stato possibile accedere ai profili e ai carrelli di altri utenti, anche senza autorizzazione esplicita.

A livello software alcune contromisure consigliate potrebbero essere:

- validazione dei token JWT e rotazione periodica delle chiavi;
- implementazione di controlli di accesso basati sui ruoli (RBAC);
- logging e monitoraggio delle attività sospette;
- protezione delle API con autenticazione forte e autorizzazione granulare.

Attacchi Realistici

Considerando che Juice Shop è un'applicazione deliberatamente vulnerabile utilizzata a scopo didattico, abbiamo deciso di riportare alcuni scenari di attacco realistici che potrebbero verificarsi in un contesto reale, qualora un software presentasse vulnerabilità simili:

- **Phishing:** un attaccante può sfruttare vulnerabilità di tipo XSS (Cross-Site Scripting) per iniettare un form di login falso all'interno dell'interfaccia dell'applicazione;
- **Account Takeover:** attraverso tecniche di user enumeration e l'utilizzo di domande di sicurezza deboli o prevedibili, è possibile avviare un processo di reset della password e ottenere il controllo completo dell'account di un altro utente;
- **Data Breach:** si tratta di un attacco mirato che può portare al dump completo del database utenti, esponendo informazioni altamente sensibili come e-mail, password, numeri di carte di credito e indirizzi;
- **Furto d'Identità:** una volta ottenuti i dati personali, possono essere utilizzati per impersonare la vittima in contesti legali, finanziari o sociali;
- **Frodi Finanziarie:** le informazioni relative a carte di credito o conti bancari possono essere utilizzate per effettuare transazioni non autorizzate, acquisti fraudolenti o trasferimenti di denaro;
- **Vendita di Dati sul Dark Web:** i dati raccolti possono essere aggregati e rivenduti su marketplace illegali nel dark web, dove hanno valore economico.

Normative

Le vulnerabilità riscontrate nella fase di Privilege Escalation costituiscono gravi violazioni delle normative europee e internazionali.

Il **GDPR** è un regolamento dell'Unione Europea che disciplina il trattamento dei dati personali delle persone fisiche all'interno dell'UE: Juice Shop viola la protezione dei dati personali a causa dell'assenza di misure tecniche adeguate e alla loro libera esposizione.

La **Direttiva NIS2**, è una normativa europea che mira a rafforzare la sicurezza cibernetica delle infrastrutture critiche e di altri settori importanti nell'Unione Europea: la gestione inadeguata degli accessi e la mancanza di resilienza del sistema rendono Juice Shop non conforme.

Lo **standard ISO 27001** sancisce i requisiti per un Sistema di Gestione della Sicurezza delle Informazioni; anche questo è violato nei controlli all'autenticazione, al principio del "Zero Trust" e al monitoraggio delle attività.

La presenza dei dati di pagamento non protetti contrasta con i requisiti del **PCI DSS**, standard di sicurezza globale per le aziende che gestiscono dati di carte di pagamento.

In un ambiente reale, queste vulnerabilità esporrebbero l'organizzazione a sanzioni legali, danni reputazionali e perdita di fiducia da parte degli utenti.

Conclusioni

La VAPT condotta sull'applicazione Juice Shop ha permesso di simulare una serie di attacchi atti a compromettere la sicurezza e l'affidabilità che, come abbiamo potuto dedurre, sono praticamente inesistenti.

Nelle varie fasi che sono state svolte, ci siamo immersi nell'attività di Red Teaming a 360 gradi:

1. **Information Gathering:** abbiamo raccolto tutte le informazioni necessarie, sia per avere una panoramica generale dell'applicazione, sia per sapere a cosa saremmo andati incontro durante le varie fasi e tipologie di attacco;
2. **Vulnerability Research e POC:** abbiamo ricercato tutte le possibili vulnerabilità e costruito le relative POC, che sono poi state attentamente verificate;
3. **Exploitation:** dopo un'attenta fase di priorizzazione, abbiamo simulato un attacco per ogni vulnerabilità da noi individuata e confermata;
4. **Post-Exploitation:** abbiamo eseguito tutte le fasi necessarie per ricavare quante più informazioni possibili dall'interno del sito e sfruttarle in maniere differenti.

Ognuna di queste fasi si è rivelata fondamentale per il procedimento successivo e per il raggiungimento di alcune importanti conclusioni: tutte queste fasi racchiudono un insieme di informazioni che, se sfruttate da un hacker con intenzioni malevoli, possono sfociare in problemi gravi, sia per quanto riguarda il sito, sia per gli utenti che lo utilizzano abitualmente.

È inoltre evidente la violazione di alcune importanti normative europee, tra cui **GDPR**, la **Direttiva NIS2** e lo **Standard ISO 27001**, fondamentali per la protezione dei dati personali e per la sicurezza informatica. Quello che ci sentiamo di consigliare, rimanendo nel pieno spirito di un'attività di Red Teaming, sono alcune raccomandazioni che permetterebbero di introdurre le misure necessarie a mantenere la sicurezza, senza un cambio eccessivo del sito: una semplice sanificazione, maggiori controlli lato-server, protezione delle API, gestione corretta dei token, monitoraggio di accessi e attività anomale...

OWASP Juice Shop si è dimostrata una piattaforma perfetta per chi è alle prime armi come noi, in quanto siamo riusciti a comprendere tutte queste dinamiche e anche a mettere in pratica alcuni attacchi: abbiamo avuto un'esperienza totalizzante e consapevole, fondamentale per il nostro futuro.

Grazie per l'attenzione.

Caroli Sofia - Passini Anna - Ricci Gabriele