Introduzione

L'obiettivo del PT è trovare e utilizzare vulnerabilità all'interno dell'applicativo web OWASP Juice Shop.

Oltre a questo report verranno allegati i file di output dei vari tool utilizzati.

Metodologia

Di seguito descriviamo step-by-step le fasi del penetration test

Information Gathering

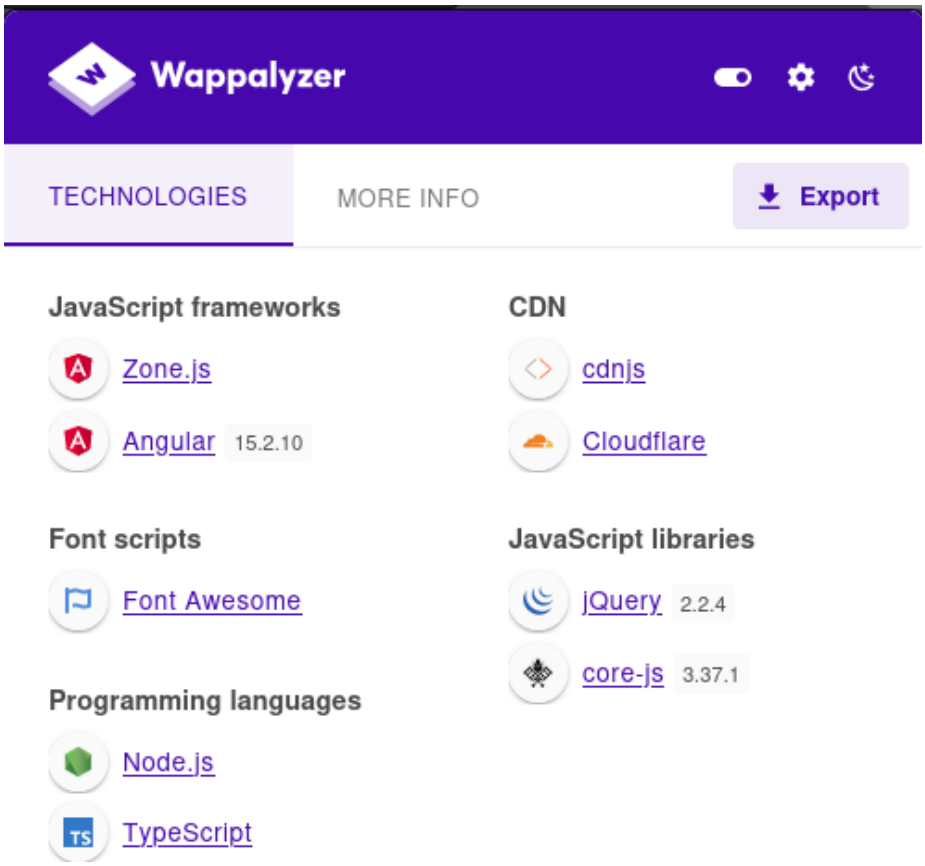
Tecniche e strumenti utilizzati per raccogliere informazioni iniziali.

Abbiamo iniziato utilizzando Nmap per fare Service Enumeration, specificando l'ip della web app e la porta con il comando:

sudo nmap -p3000 -sV -O 172.17.0.2

senza però ottenere risultati utili (file: nmapOutput.txt).

Per cercare di trovare qualche tecnologia utilizzata (con relative versioni) abbiamo utilizzato Wappalyzer, ottenendo:

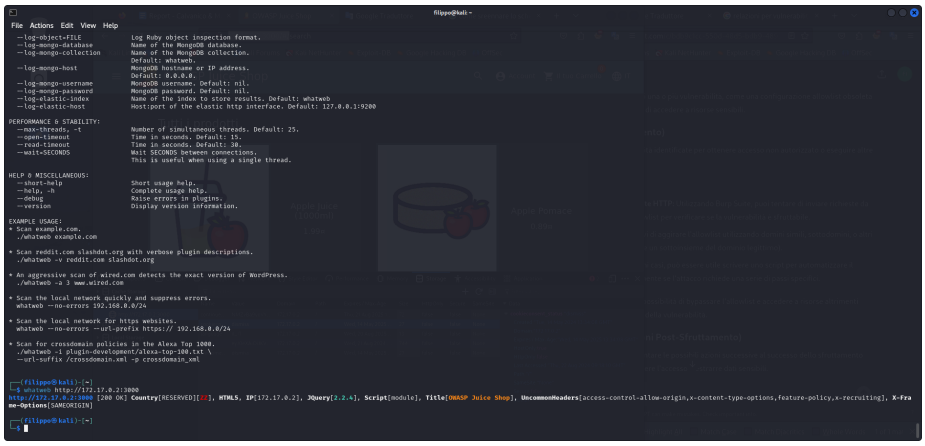


Dal tool otteniamo solo 3 versioni, e cercando nel web vulnerabilità specifiche troviamo:

- Angular 15.2.10: nessuna vulnerabilità;
- jQuery 2.2.4: vulnerabile ad attacchi di tipo Cross-site Scripting (XSS);
- core-js 3.37.1: nessuna vulnerabilità.

In sostanza terremo utile questa unica informazione per le fasi successive.

Un altro tool utilizzabile per scovare informazioni sulla web app è WhatWeb:

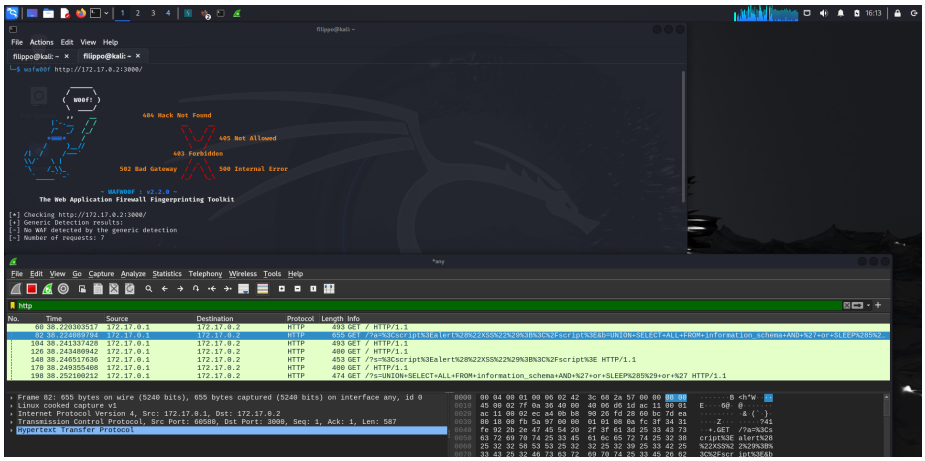


Il passo successivo è controllare, tramite WAFW00F, se l'applicazione ha un WAF (Web Application Firewall), e tramite il comando:

wafw00f http://172.17.0.2:3000

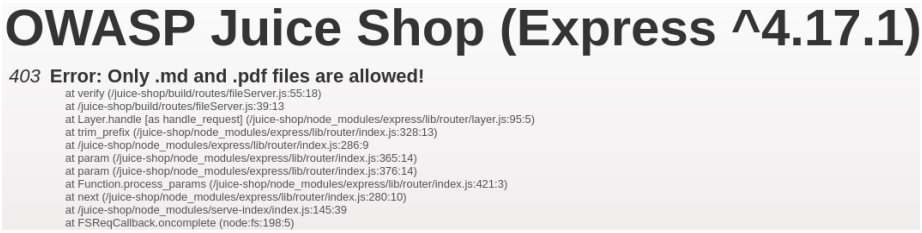
abbiamo ottenuto una risposta negativa che quindi ci conferma che non è presente un WAF (file: outputWafw00f.txt).

Per avere un ulteriore conferma che le risposte http non vengano alterate e capire se c'è uno strumento di detection/response tramite l'utilizzo di Wireshark abbiamo filtrato le risposte HTTP, notando che la prima richiesta fatta è classica, per poi andare ad eseguire altre richieste con dei failed, ad esempio "Ealert".



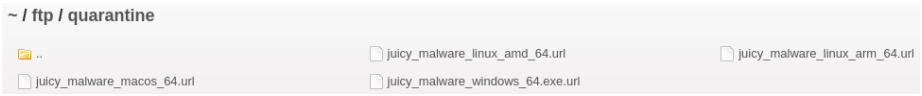
Per cercare alcuni path o file interessanti abbiamo usato Nikto, ottenendo un gran numero di falsi positivi ma anche risultati (file: outputNikto), in particolare:

- /ftp: contiene molti file non leggibili, ma tra quelli leggibili abbiamo "acquisitions.md" un file confidenziale. L'apertura dei file non leggibili ci permette di scoprire, tramite errore, che la web app usa Express 4.17.1;



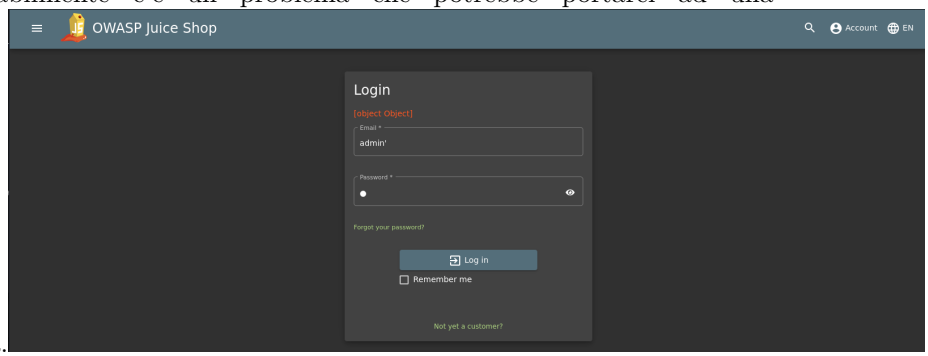
Vulnerabile ad attacchi di tipo Open Redirect

- /ftp/quarantine: directory con malware per diversi OS;



Successivamente siamo passati alla pagina di login cercando, in qualche modo, di provocare errori che ci avrebbero dato indizi sulle tecnologie o metodologie utilizzate nel backend.

Inserendo caratteri inaspettati dalla form, come per esempio il singolo apice, abbiamo ottenuto un errore diverso dal solito, che ci fa capire come probabilmente c'è un problema che potrebbe portarci ad una vulnerabilità.



Nella parte di vulnerability assessment indagheremo meglio.

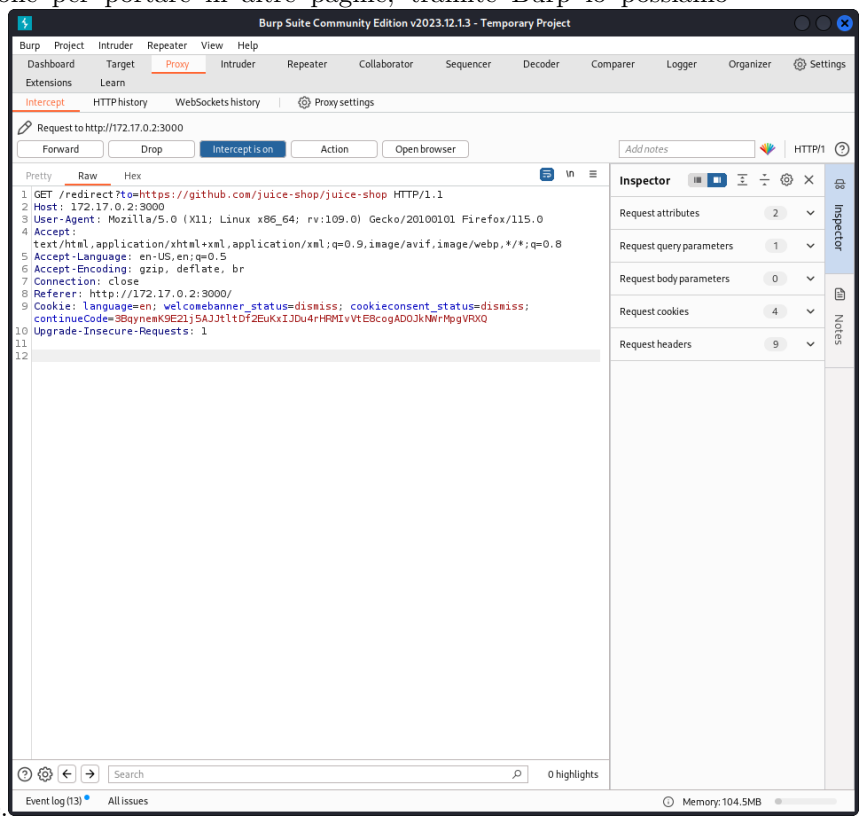
Il passo successivo è stato quello di trovare nuove directory e sottodirectory tramite tool automatizzati, nel nostro caso Ffuf (file: outputFfuf), dove tramite comando:

ffuf -u http://172.17.0.2:3000/FUZZ -w /usr/share/dirbuster/wordlists/directory-list-1.0.txt -fs 3740-3750

abbiamo cercato all'interno della web app tutti i possibili percorsi, passandogli una wordlist (contenuta nativamente dentro kali) con parole chiave da cercare e togliendo tutti i risultati che ritornavano una size compresa tra quella inserita (la size della homepage); questo perché il sito ritorna la pagina iniziale ogni volta che si prova ad accedere ad un indirizzo inesistente.

Tra i path trovati quelli effettivamente raggiungibili e interessanti sono:

- /restoration: path che ritorna un errore e come i precedenti ci da informazioni sul backend;
- /apictureofbritain: come il precedente
- /video: ci mostra un video altrimenti non raggiungibile;
- /metrics: un'insieme di metriche di nodejs;
- /redirect: da questo path se aggiunto ?to= è possibile arrivare a parti altrimenti non raggiungibili, questo perchè i bottoni utilizzano questa combinazione per portare in altre pagine, tramite Burp lo possiamo dimostrare.



Notando che alcune pagine contenevano nel path: /#/nome abbiamo pensato di fare nuove scansioni (file: outputFfuf2.txt), sempre con fuff però usando un link diverso e con delle wordlist differenti (relative al tool):

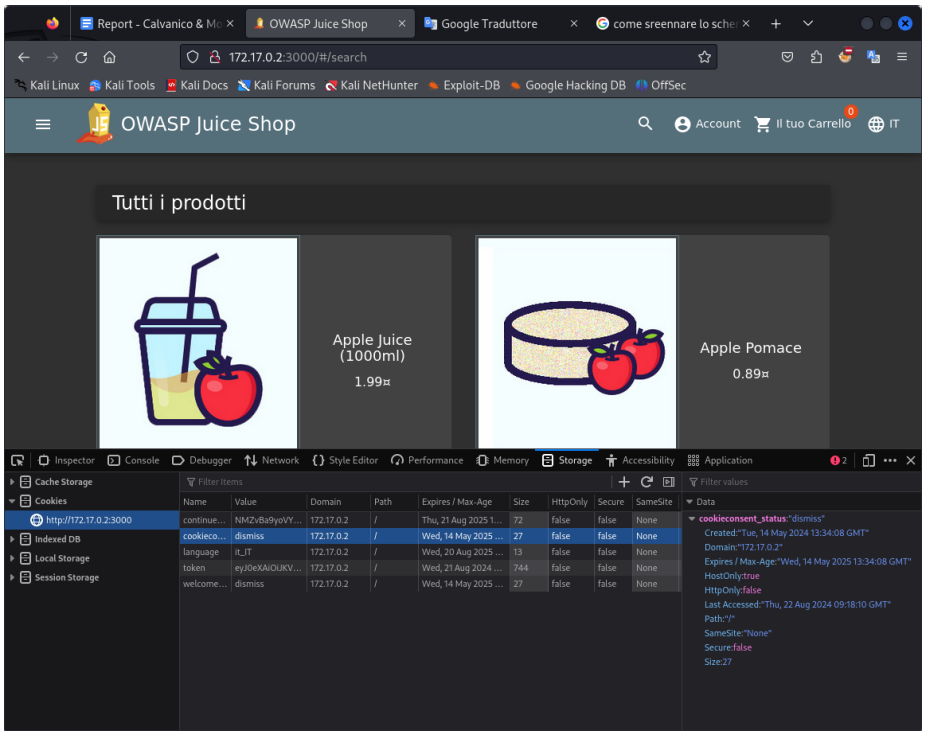
ffuf -u http://172.17.0.2:3000/#/FUZZ -w /usr/share/wfuzz/wordlists/common.txt -o outputFfuf2.txt

In questo caso abbiamo tolto il filtro perché con esso non dava risultati.

Togliendo il filtro otteniamo molti falsi positivi, ma dopo aver controllato alcuni path interessanti abbiamo trovato:

- /administration: il path ritorna errore 403, questo vuol dire che esiste ma non è accessibile senza permessi, sarà da ricontrollare dopo aver fatto l'accesso;

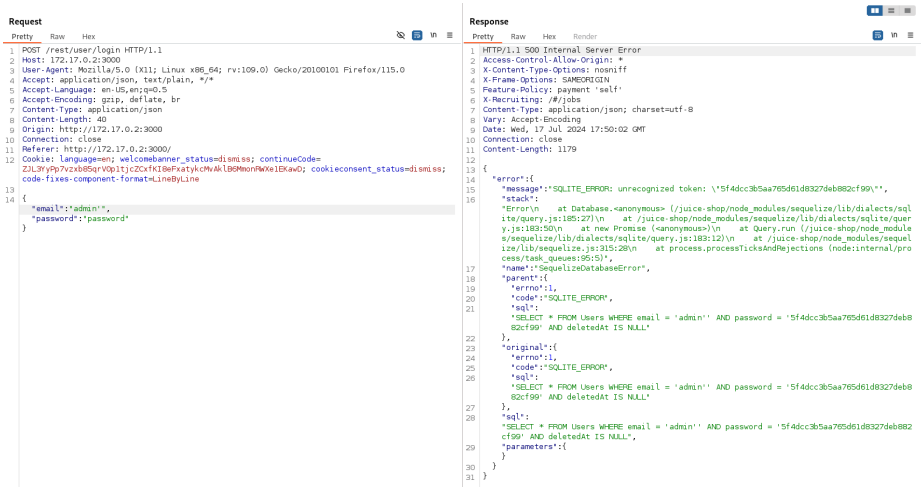
Infine controlliamo anche i cookie per cercare nei flag eventuali parametri non conformi da utilizzare per scovare vulnerabilità, senza però riscontrare nulla di anomalo.



Vulnerability Assessment

Strumenti e metodi per identificare le vulnerabilità.

Abbiamo iniziato la fase indagando l'errore anomalo ottenuto dalla form, usando il Repeater di Burp, per modificare le richieste di log in maniera più veloce e per ottenere le risposte complete, abbiamo inserito dei caratteri inaspettati generando questo errore:



andando a scoprire esattamente la query utilizzata e che viene utilizzato SQLite.

In questo modo potremmo fare un attacco di tipo sql injection per poter accedere con un account.

Exploitation

Tentativi di sfruttamento delle vulnerabilità trovate.

Dopo aver riscontrato nella fase precedente che la vulnerabilità nel login esiste abbiamo eseguito un attacco di tipo SQL Injection, andando ad inserire nel campo email:

admin' or 1=1 –

permettendoci di accedere con l'account admin, questo perché l'email verrà sempre considerata vera (grazie al 1=1) e la password verrà saltata, andando a prendere il primo account salvato del DB.

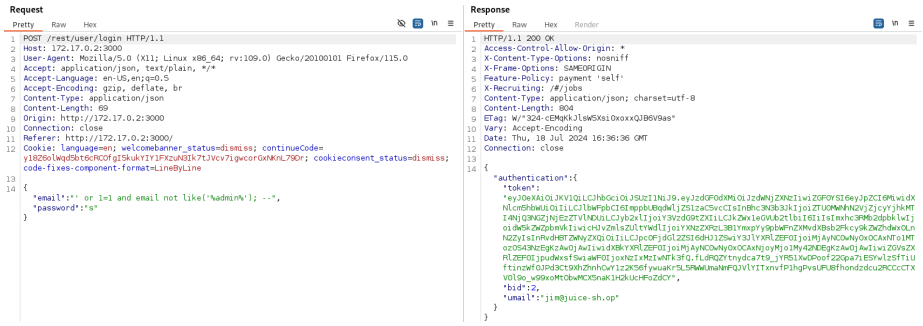
Con questa informazione, e una conoscenza pregressa di SQL, abbiamo provato a trovare altri account con cui accedere utilizzando un altro mezzo messo a disposizione da SQL stesso: il LIKE.

Nel dettaglio abbiamo impostato un attacco SQL Injection inserendo nella form:

admin' or 1=1 and email not like('%admin%'); –

[I % contano come segnaposto all'interno di SQL]

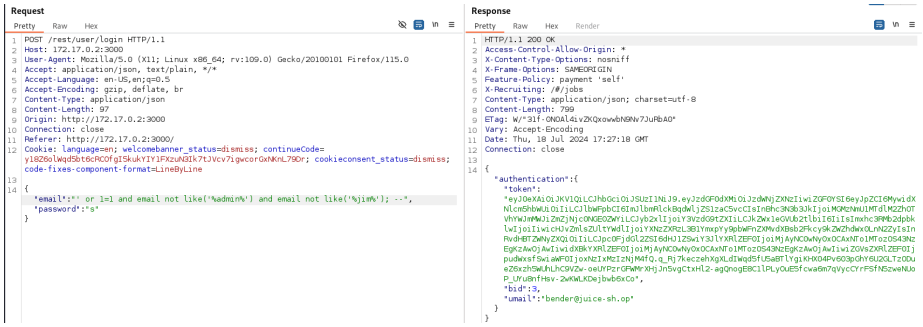
in questo modo otterremo accesso all'account successivo, che in questo caso è Jim:



Continuando su questa strada abbiamo fatto:

admin' or 1=1 and email not like('%admin%') and email not like('%jim%'); –

ottenendo accesso ad un altro account denominato Bender:



Andando avanti così è possibile accedere a tutti gli account esistenti, creando non pochi problemi.

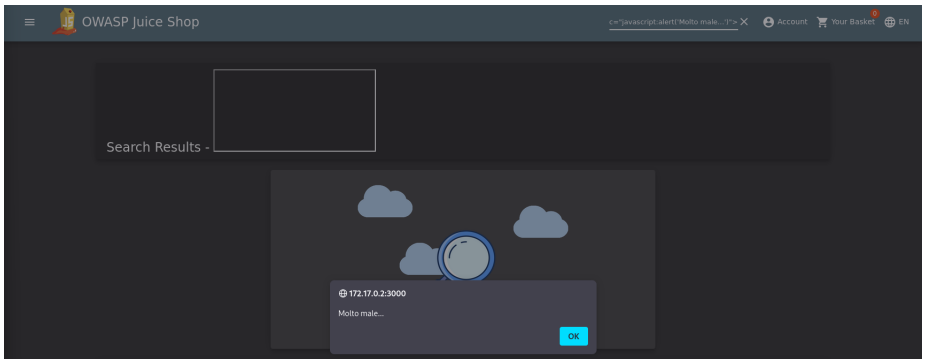
Ricordandoci dalla fase di information gathering che il sito usa jQuery versione 2.2.4 sappiamo che è vulnerabile ad attacchi di tipo Cross-site Scripting (XSS).

Tramite questa informazione possiamo provare ad inserire codice javascript e tag html in tutte quelle parti che accettano input dall'utente.

Se per esempio nella barra di ricerca mettiamo tag html e/o codice javascript questo viene inserito senza nessun tipo di controllo, ottenendo questi risultati:

Questo inserendo <button>Click Me!</button>.

Oppure inserendo <iframe src="javascript:alert('Molto male...')">:

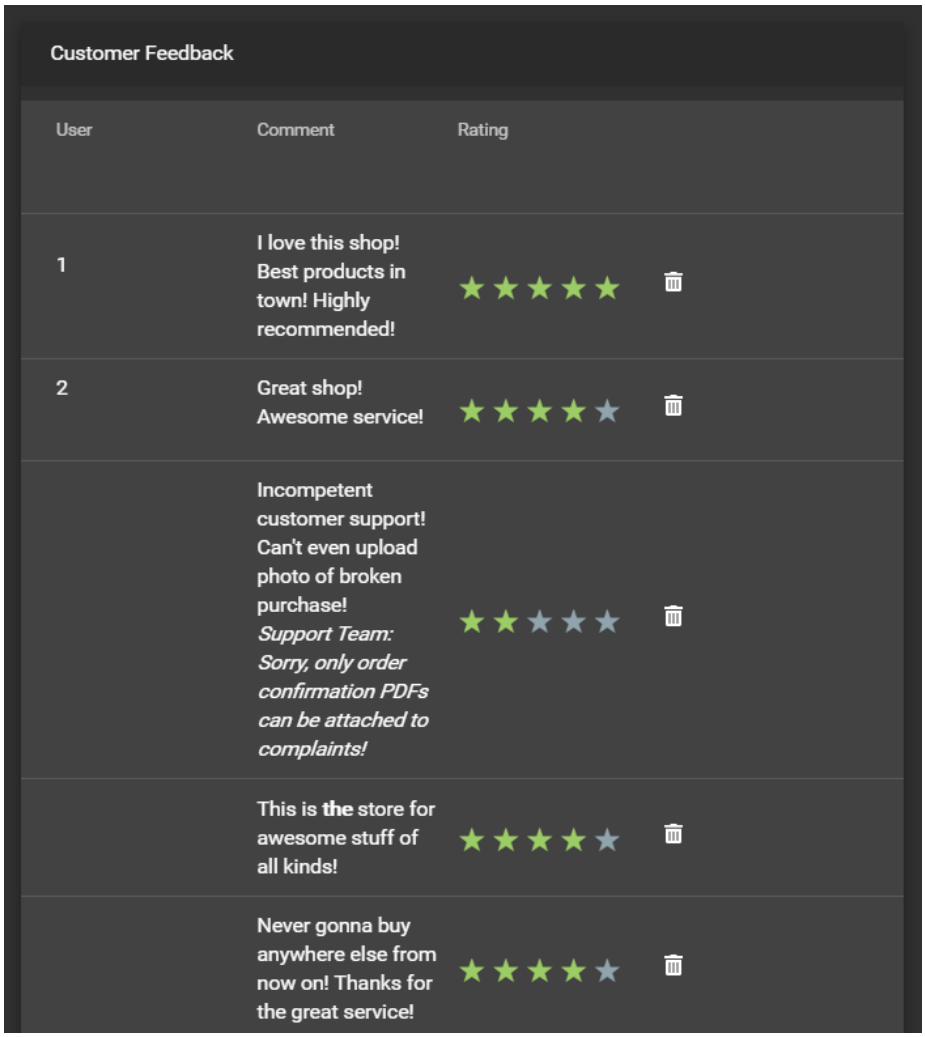


Post-Exploitation

Azioni compiute dopo l'accesso iniziale.

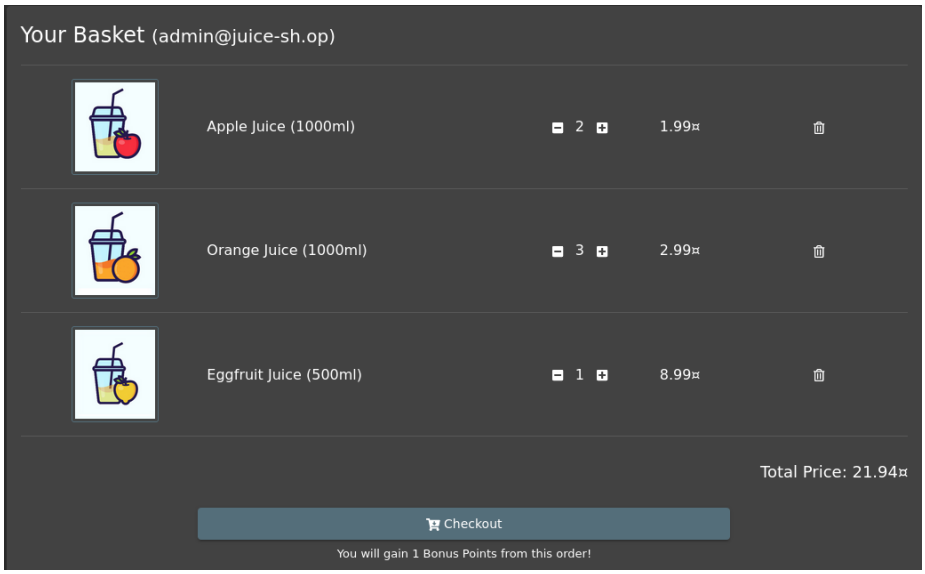
Dopo aver ottenuto l'accesso all'account admin siamo andati a controllare il path trovato nella fase di Information Gathering, /administration, che era inaccessibile senza permessi.

In questa sezione possiamo vedere tutti gli account e anche tutte le recensioni rilasciate, con la possibilità di rimuoverle.



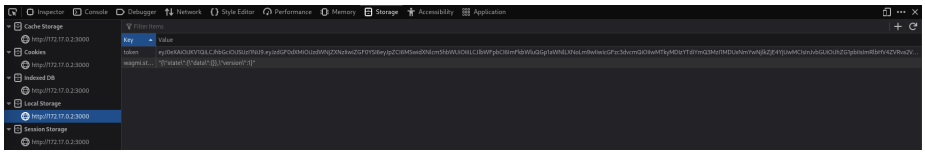
Successivamente abbiamo iniziato a controllare le varie pagine disponibili solo ad un account registrato.

La nostra attenzione è passata subito alla sezione “Your Basket”, che nell’account admin conteneva già dei prodotti:



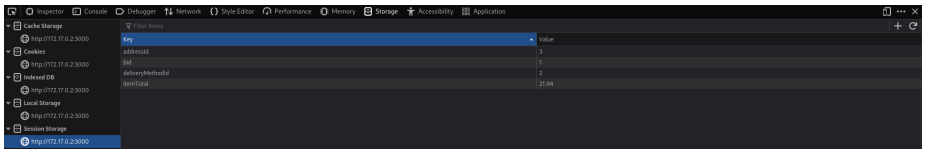
Andando a controllare come questi dati vengono salvati anche alla chiusura della pagina o al cambio di schermata abbiamo aperto la sezione Storage nello strumento degli sviluppatori del browser; questo grazie alle conoscenze pregresse ottenute durante un altro corso, dove abbiamo creato anche noi una web app per ordinare prodotti.

Nello specifico abbiamo controllato il Local e Session Storage:



Il Local contiene il token di autenticazione.

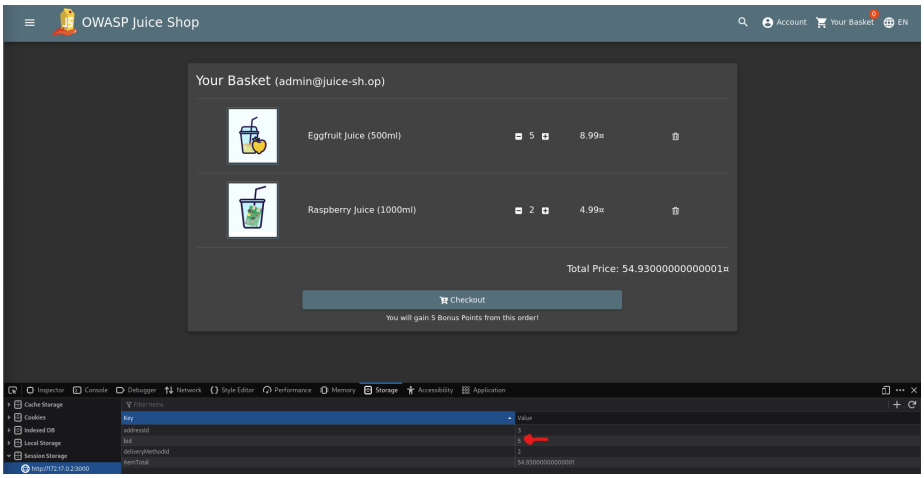
Invece il Session i dettagli del carrello:



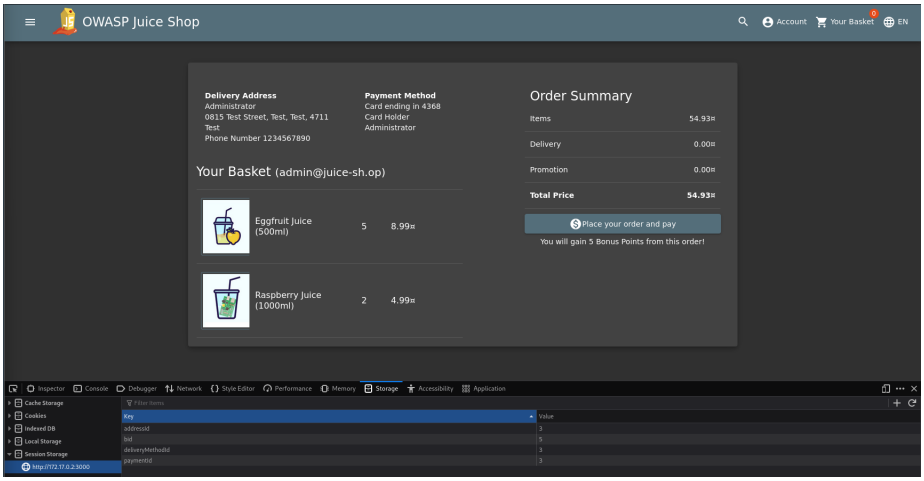
I due valori che ci sono subito caduti sott'occhio sono il prezzo totale (itemTotal) e l'id del carrello (bid).

Andando a modificare il primo a 0 per provare a non pagare i prodotti notiamo, come giusto che sia, che il valore viene ricalcolato rendendo impossibile questo trucco.

Modificando invece il bid possiamo accedere ad altri carrelli:



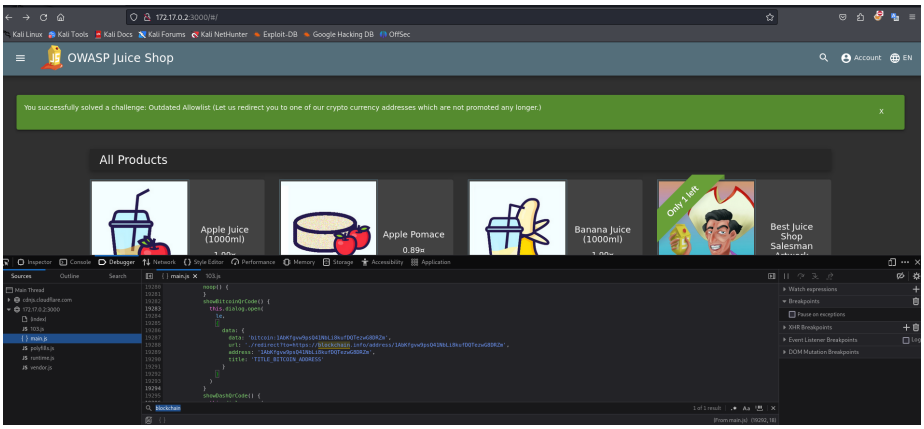
Riuscendo anche a fare il checkout:



Sapendo dalla fase di Information Gathering che la web app utilizza il "redirect?to" per reindirizzare verso altre pagine, abbiamo deciso di fare una ricerca all'interno del file main.js per trovare tutti gli utilizzi.

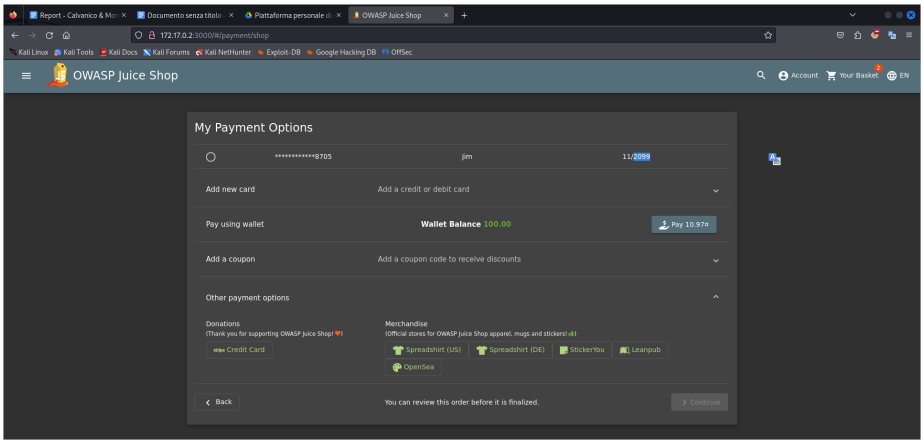
Utilizzando il dev tool del browser abbiamo cercato tramite Ctrl+F tutte le occorrenze del redirect nel file di script, trovando un redirect che normalmente non sarebbe accessibile.

L'URL ritorna a un crypto wallet che probabilmente veniva utilizzato per ricevere



donazioni.

Andando a ritroso cercando dove venga utilizzata la funzione showBitcoinQrCode possiamo notare che viene chiamata nella parte di gestione del carrello, ma nella web app non è presente.



Strumenti Utilizzati

Burp Suite

Motivo dell'utilizzo

Lo strumento ci aiuta con un Proxy per intercettare e successivamente visualizzare le richieste della web app.

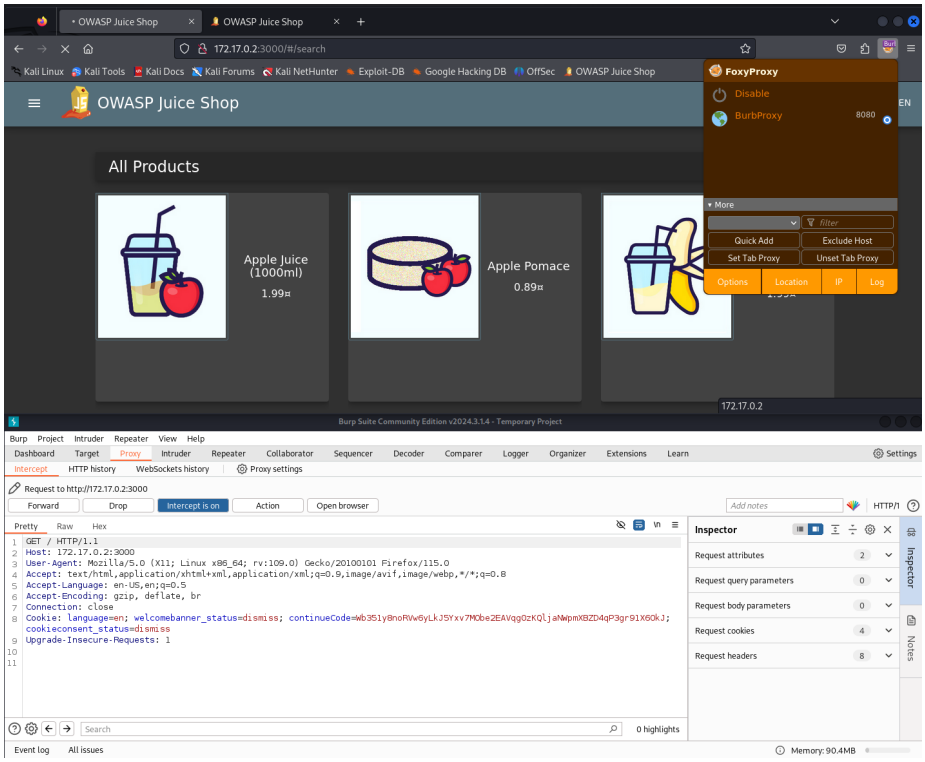
A differenza degli strumenti degli sviluppatori integrati nel browser, Burp è più flessibile, permettendo di modificare le richieste, visualizzarle nella sezione Logger, ecc.

Obiettivo della scansione

L'obiettivo è cercare nelle richieste informazioni utili per individuare vulnerabilità ed effettuare exploit.

Spiegazione del funzionamento

Burp intercetta e blocca le richieste verso e da la web app, questo è possibile perché abbiamo installato un'estensione nel browser utilizzato (FireFox), chiamata FoxyProxy, che implementa un proxy che ri-indirizza le richieste verso Burp permettendoci di interagire con esse.



Nmap

Motivo dell'utilizzo

Utilizzato per fare service enumeration, specifichiamo l'ip + porta e il tool ci restituisce i servizi, con rispettive versioni utilizzate.

Obiettivo della scansione

Scoprire che servizi vengono utilizzati dalla web app con anche la versione per

cercare possibili vulnerabilità.

Spiegazione del funzionamento

Tramite comando:

nmap -p[PORT] [FLAG] [IP]

il tool invia pacchetti al host e in base alla risposta capisce se una porta è aperta, filtrata o chiusa e quindi se un servizio è utilizzato o meno oppure può ottenere informazioni sul OS.

Wappalyzer

Motivo dell'utilizzo

Utilizzato per ottenere informazioni sulle tecnologie utilizzate dal sito e relative versioni

Obiettivo della scansione

Trovare tecnologie vulnerabili nella web app in modo da fare l'exploit su di esse.

Spiegazione del funzionamento

il tool cerca negli header http, nel codice html e script informazioni che potrebbero ritornare a certe tecnologie e versioni.

WAF00F

Motivo dell'utilizzo

Tool perfetto per controllare la presenza di WAF in maniera veloce e senza troppe configurazioni.

Obiettivo della scansione

Trovare che tipo di WAF viene usato in modo da sapere quale tipo di traffico viene bloccato o meno, utile per usare certi exploit.

Spiegazione del funzionamento

Tramite comando:

waf00f [INDIRIZZO]

invia una serie di richieste HTTP a un sito web e analizza le risposte per identificare il WAF in uso.

Wireshark

Motivo dell'utilizzo

Fare lo sniff dei pacchetti HTTP verso e dalla web app.

Obiettivo della scansione

Prova manuale per avere un'ulteriore conferma che le risposte HTTP non vengano alterate da uno strumento di detection/response.

Spiegazione del funzionamento

Il tool intercetta i pacchetti in arrivo e verso un obiettivo con la possibilità di filtrarli.

Nikto

Motivo dell'utilizzo

Come automatic scanners è veloce e applicabile a qualsiasi realtà.

Obiettivo della scansione

Trovare path che non dovrebbero essere accessibili dagli utenti in modo da scovare informazioni sensibili.

Spiegazione del funzionamento

Tramite comando:

nikto -host[INDIRIZZO]

il tool ricerca informazioni sul web server o sull'applicazione web da scansionare; successivamente inizia la scansione utilizzando richieste http e file.

FFUF

Motivo dell'utilizzo

Abbiamo scelto questo tool invece che altri, con lo stesso scopo, perché più veloce e con tanti flag utili per personalizzare la ricerca.

Obiettivo della scansione

Trovare nuove sottodirectory per scovare vulnerabilità o tecnologie usate dalla web app.

Spiegazione del funzionamento

Tramite comando:

ffuf -u[INDIRIZZO] -w[PATH WORDLIST] -fs[SIZE RISPOSTA]

il tool parte dal rootdir e ricerca ricorsivamente su ogni sottodirectory trovata; i due flag utilizzati fanno:

- -w: prende una wordlist dal path inserito;
- -fs: filtra le risposte in base al size del ritorno, ci è servito perché la webapp in caso di indirizzo errato ritorna la pagina iniziale.

Analisi delle Vulnerabilità

SQL Injection - Login

Descrizione della vulnerabilità

Tramite l'inserimento di certi caratteri e stringhe tipiche di SQL, e inaspettati dal server, è possibile eseguire codice malevolo.

In questo caso usata per ottenere accesso ad uno o più account (tra cui quello dell'admin).

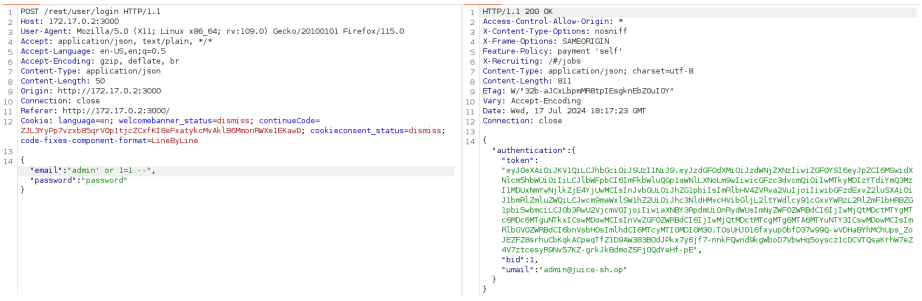
Riproducibilità

Per riprodurre la vulnerabilità bisogna:

1. andare nella pagine di login;
2. inserire la seguente stringa nell'email: Placeholder' or 1=1 –;
3. inserire qualunque cosa nella password;
4. cliccare invio.

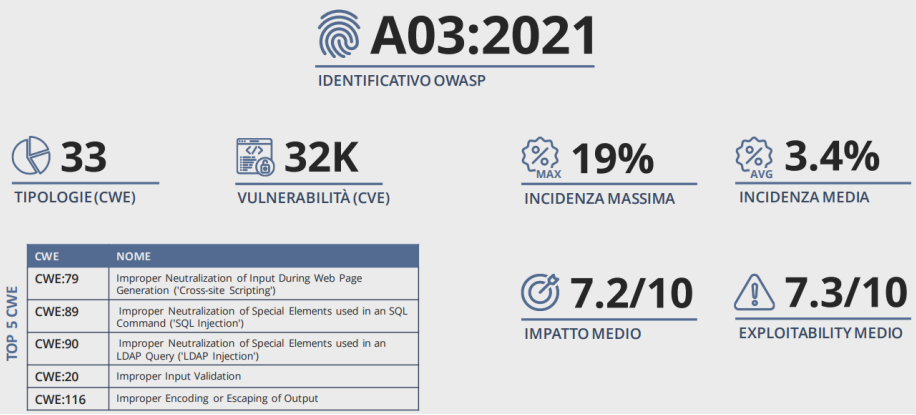
In sostanza va messo dopo l'or una condizione sempre vera, seguito dai simboli per commentare.

Prova della rilevazione

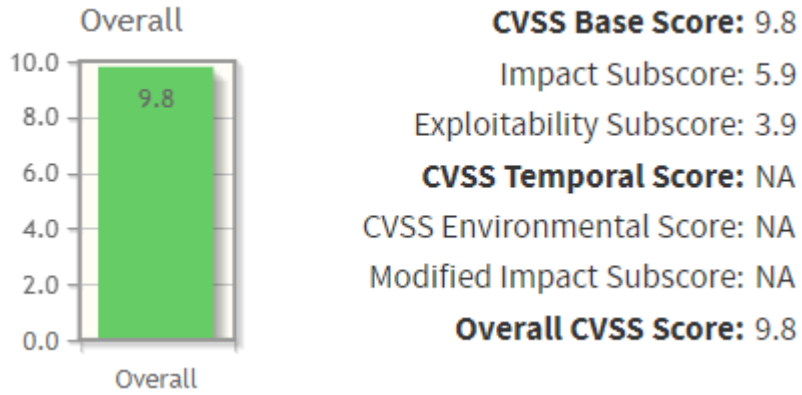


Classificazione OWASP TOP 10

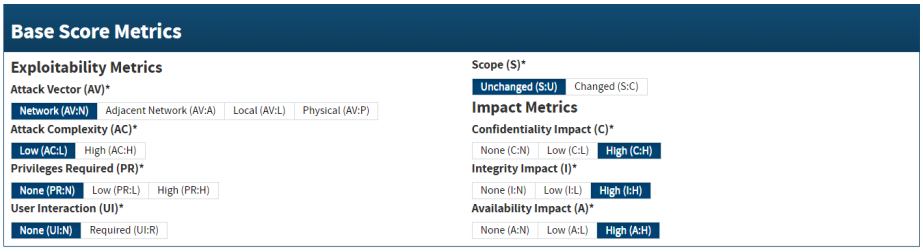
Secondo l'OWASP questo tipo di vulnerabilità è classificata:



Lo score del CVSS è:



Con le seguente metriche:



Requisiti dell'attaccante

L'unica cosa necessaria è avere accesso al sito web e una conoscenza di SQL.

Gravità e Impatti

Questa vulnerabilità è estremamente grave, perché permette di accedere come admin all'applicativo web con un impatto non indifferente sull'integrità dell'applicativo e dei dati.

XSS - DOM

Descrizione della vulnerabilità

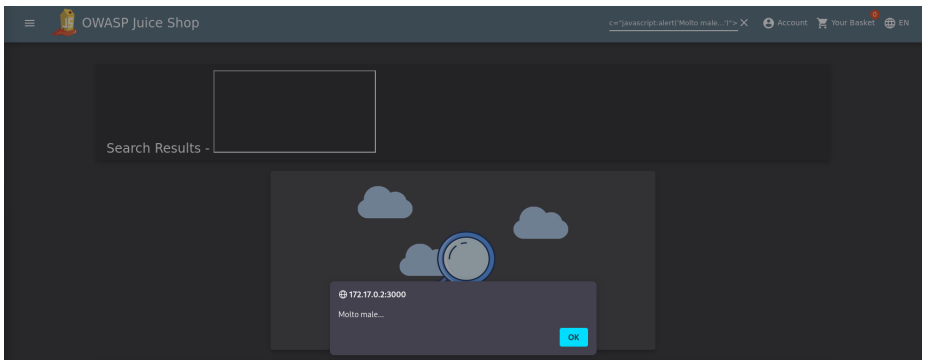
Un Cross-Site Scripting è una vulnerabilità che permette a un malintenzionato di iniettare codice dannoso all'interno di un sito web legittimo.

Riproducibilità

Nel nostro caso tramite il campo ricerca siamo riusciti ad inserire nel DOM componenti HTML, come bottoni e alert, ecco gli step:

1. aprire il campo ricerca;
2. inserire qualunque tag si voglia, nel nostro caso: <iframe src="javascript:alert(‘Molto male. . . ‘)">;
3. all'invio si vedrà nel DOM della pagina il tag inserito nel campo di input.

Prova della rilevazione



Classificazione OWASP TOP 10

Secondo l'OWASP questo tipo di vulnerabilità è classificata:

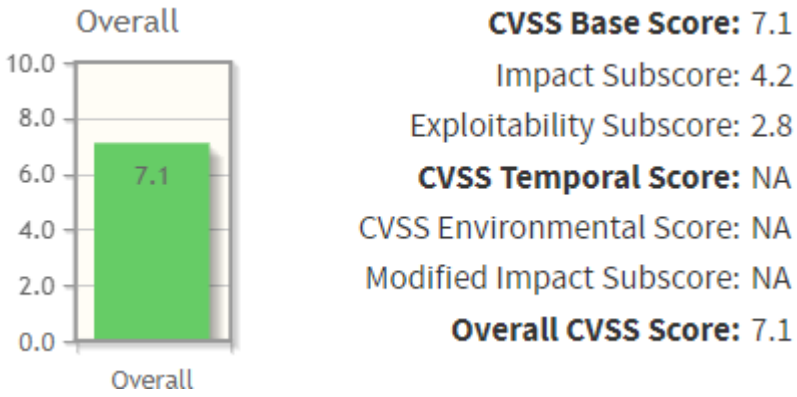
A7:2017-Cross-Site Scripting (XSS)

Languages: [en] de

← A6:2017-Security Misconfiguration | OWASP Top Ten 2017 PDF version | A8:2017-Insecure Deserialization →

Threat Agents / Attack Vectors		Security Weakness		Impacts	
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.		XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications. Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.		The impact of XSS is moderate for reflected and DOM XSS, and severe for stored XSS, with remote code execution on the victim's browser, such as stealing credentials, sessions, or delivering malware to the victim.	

Lo score del CVSS è:



Con le seguente metriche:

Base Score Metrics

Exploitability Metrics

Attack Vector (AV)*
Network (AV:N) | Adjacent Network (AV:A) | Local (AV:L) | Physical (AV:P)

Attack Complexity (AC)*
Low (AC:L) | High (AC:H)

Privileges Required (PR)*
None (PR:N) | Low (PR:L) | High (PR:H)

User Interaction (UI)*
None (UI:N) | Required (UI:R)

Scope (S)*
Unchanged (S:U) | Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*
None (C:N) | Low (C:L) | High (C:H)

Integrity Impact (I)*
None (I:N) | Low (I:L) | High (I:H)

Availability Impact (A)*
None (A:N) | Low (A:L) | High (A:H)

Requisiti dell'attaccante

L'attaccante deve solo aver accesso alla web app e conoscere un minimo di html.

Gravità e Impatti

Se la modifica, che viene fatta tramite questa vulnerabilità, rimane salvata all'interno della pagina il codice javascript verrà eseguito sul browser dell'utente ad ogni accesso, creando situazioni altamente pericolose.

La vulnerabilità ha una gravità moderata se il codice non viene salvato (Reflected

XSS) con impatti medio/bassi, al contrario si ha una gravità e un impatto critici.

Broken Access Control - Accesso ad un altro carrello

Descrizione della vulnerabilità

Tramite una configurazione errata del Session Storage e dei permessi è possibile accedere al carrello di un altro utente, con la possibilità di effettuare tutte le normali attività.

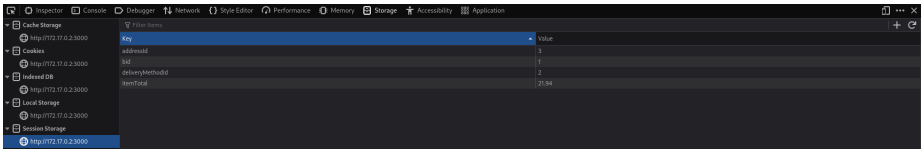
Riproducibilità

Per effettuare questa vulnerabilità è necessario:

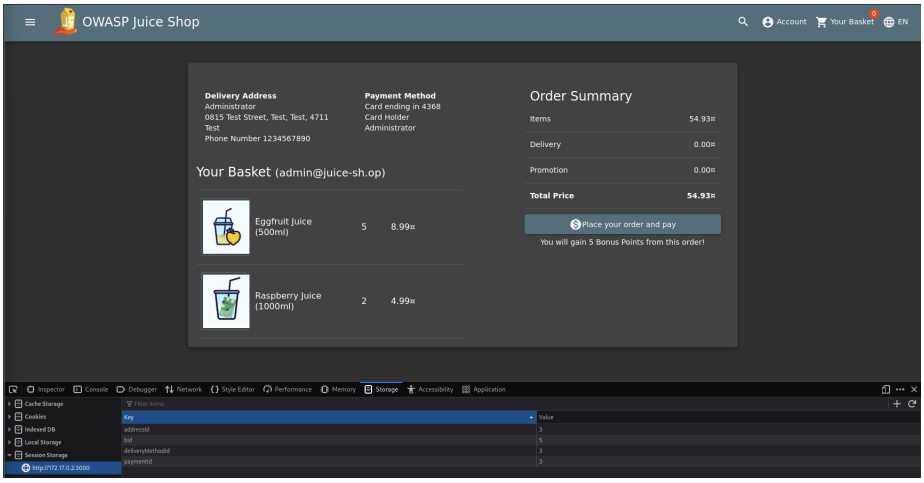
1. essere autenticati ad un account;
2. aprire il carrello e il dev tool del browser;
3. nella sezione Session Storage cambiare il campo bid

Prova della rilevazione

Bid prima del cambio:

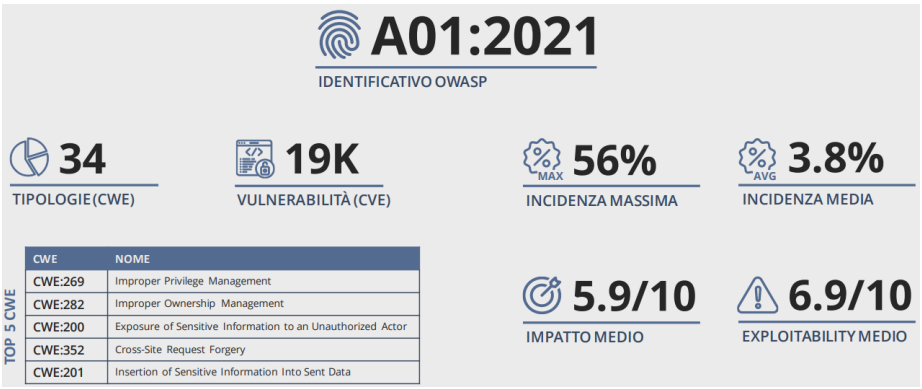


Bid e carrello dopo il cambio:

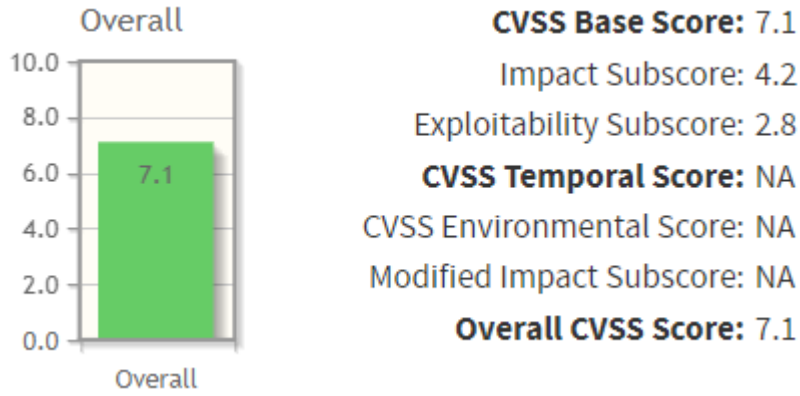


Classificazione OWASP TOP 10

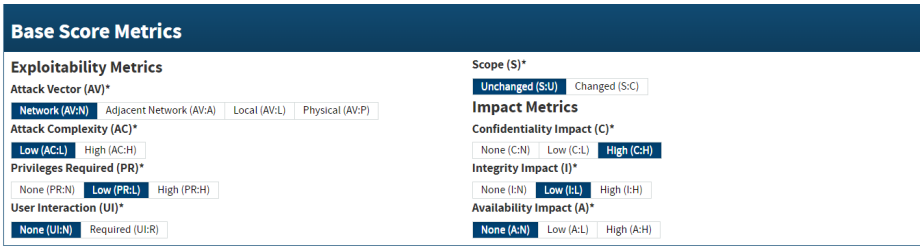
Secondo l'OWASP questo tipo di vulnerabilità è classificata:



Lo score del CVSS è:



Con le seguente metriche:



Requisiti dell'attaccante

L'attaccante deve avere un account con cui accedere al proprio carrello e saper utilizzare il dev tool.

Gravità e Impatti

L'attaccante non può fare molti danni perché riesce ad accedere solo a quali prodotti l'utente ha nel carrello, portando ad avere un impatto sulla confidenzialità con una gravità abbastanza bassa.