SCRIVERE CODICE SICURO

Febbraio 2024

SCRIVERE CODICE SICURO

Programma

1) **PARTE PRIMA** - Introduzione ai più comuni attacchi ad un sito/applicativo web
   a. Cross site scripting (XSS)
   b. Remote code injection
   c. SQL Injection
   d. Attacchi al File System
2) **PARTE SECONDA** – cosa fare lato client
   a. Securizzare un form in HTML
   b. Securizzare un form in Javascript
   c. Cosa è OWASP (Open Web Application Security Project)
3) **PARTE TERZA** –
   a. Prevenire le SQL Injection
   b. Proteggere UPLOAD dei files
   c. Proteggere i dati / limitare i tentativi di accesso

PARTE PRIMA

- Cross Site Scripting XSS
- Remote Code Injection
- SQL Injection
- Attacchi al File System

CROSS SITE SCRIPTING - XSS

Il Cross Site Scripting (conosciuto anche come HTML Injection), è quell'attacco che permette l'esecuzione di codice JavaScript (o altro codice interpretabile dal client) **non verificato** e non previsto dall'applicazione oppure l'inserimento di contenuto non previsto all'interno delle proprie pagine HTML.

Questo codice viene solitamente fatto eseguire aggiungendo dei tag HTML/JS in posizioni in cui questi non erano previsti.

In PHP è possibile eseguire attacchi XSS in tre modi differenti:

In primo luogo l'aggressore può mostrare un link che rimanda ad un sito Internet o ad una pagina con contenuto pericoloso.

In secondo luogo è possibile che l'aggressore salvi dei dati nocivi direttamente nel database, utilizzando delle falle nel sistema di salvataggio dei dati, che vengono successivamente visualizzati alla vittima.

Infine l'aggressore potrebbe spingere all'esecuzione di codice JavaScript direttamente nel client per modificare il DOM della pagina affinché operi in modo differente da quello che ci si aspetterebbe.

CROSS SITE SCRIPTING - XSS

Purtroppo non esiste alcun sistema interno a PHP per proteggersi da questo tipo di attacchi e quindi è strettamente necessario che si operi con attenzione quando si sviluppano applicazioni che hanno a che fare con la visualizzazione dell'input immesso dall'utente.

Per proteggersi da questo tipo di attacchi è dunque necessario fare affidamento a sistemi che validino e filtrino l'input, disabilitare register_globals (php.ini), recuperare l'input dell'utente dalla locazione corretta anziché appoggiarsi sulla variabile $_REQUEST, usare ad es. un file «DISPATCHER» che accetti e ridirezioni solo le URL accettate dall'applicazione, effettuare la codifica degli URL visualizzati all'utente utilizzando urlencode ed infine validare il codice JavaScript in modo che sia immune da attacchi di DOM injection.

CROSS SITE SCRIPTING - XSS

Per controllare che un modulo venga inviato da un URL valido in PHP, puoi utilizzare la variabile **$_SERVER['HTTP_REFERER']**.

Esempio di codice…

```
<?php
// Controlla se l'URL di provenienza è presente e corrisponde all'URL atteso
if (isset($_SERVER['HTTP_REFERER']) && $_SERVER['HTTP_REFERER'] === 'https://www.example.com/pagina
    // L'URL di provenienza è valido
    // Gestisci il modulo qui
} else {
    // L'URL di provenienza non è valido
    // Esegui azioni per affrontare il tentativo di accesso non autorizzato ad esempio, reindirizza l'utente a una pagina di errore o registra l'evento di a
    header('Location: errore.php');
    exit;
}
```

CROSS SITE SCRIPTING – TIPI DI ATTACCO

Gli attacchi XSS possono essere suddivisi in tre categorie principali:

- **Stored XSS**: Il codice dannoso viene immagazzinato sul server web e viene fornito a ogni utente che visita la pagina infetta.
- **Reflected XSS**: Il codice dannoso viene fornito all'utente attraverso un link malevolo o una richiesta manipolata, e viene riflesso dal server come parte della risposta.
- **DOM-based XSS**: Il codice dannoso modifica il Document Object Model (DOM) della pagina web direttamente sul lato client, senza coinvolgere il server.

CROSS SITE SCRIPTING – XSS (COSA FARE)

VALIDARE e SANITIZZARE l'input

Il validare **verifica** che un dato sia valido o meno

Il sanitizzare **modifica** il dato per renderlo valido

CROSS SITE SCRIPTING – XSS (COSA FARE)

Validare ….

La validazione dei dati riguarda la **verifica e la convalida dei dati** inseriti dagli utenti per assicurarsi che rispettino determinati criteri o requisiti predefiniti. L'obiettivo principale della validazione dei dati **è assicurarsi che i dati siano corretti** in base a regole specifiche o formati attesi.

- La validazione dei dati viene eseguita principalmente lato client (nel browser dell'utente) e/o lato server (nel server web). Lato client, la validazione dei dati viene spesso implementata utilizzando funzionalità native dell'HTML come l'attributo "required" per i campi obbligatori o l'attributo "pattern" per specificare un'espressione regolare che definisce il formato accettato per un campo.
- Lato server, la validazione dei dati viene solitamente eseguita attraverso la logica di programmazione e i controlli eseguiti sulle richieste inviate al server. Questo può includere la verifica della presenza dei campi obbligatori, la convalida dei formati dei dati, la limitazione delle lunghezze dei campi e così via.

CROSS SITE SCRIPTING – XSS (COSA FARE)

Sanitizzare ….

- La sanitizzazione dei dati riguarda la **rimozione o la neutralizzazione** di caratteri o codice potenzialmente dannosi o non validi dai dati inseriti dagli utenti. L'obiettivo principale della sanitizzazione dei dati è proteggere l'applicazione da attacchi come l'injection di codice malevolo.
- La sanitizzazione dei dati viene eseguita principalmente lato server durante l'elaborazione dei dati inviati dai moduli web. Gli input ricevuti dagli utenti vengono elaborati e **puliti** attraverso l'applicazione di filtri o funzioni specifiche che rimuovono caratteri speciali, codice HTML o script potenzialmente dannosi. In questo modo si previene l'esecuzione involontaria di codice malevolo e si riduce il rischio di attacchi come l'XSS (Cross-Site Scripting) o l'SQL injection.

CROSS SITE SCRIPTING – XSS (COSA FARE)

Validare e Sanitizzare differenze:

Mentre la **validazione** dei dati si concentra sulla **correttezza dei dati** rispetto a regole o formati specifici, la **sanitizzazione dei dati** si concentra sulla **rimozione o la neutralizzazione di elementi potenzialmente dannosi presenti nei dati stessi**. Entrambi i processi sono essenziali per garantire l'integrità e la sicurezza dei dati nell'ambito delle applicazioni web.

CROSS SITE SCRIPTING – XSS (COSA FARE)

VALIDARE :

Ad esempio, se stai aspettando un indirizzo email, puoi utilizzare la funzione filter_var() con il filtro FILTER_VALIDATE_EMAIL per verificare se l'input è un'email valida.

Altri esempi di filtri per validare…

FILTER_VALIDATE_BOOLEAN – Restituisce true per i valori "1", "true", "on" e "yes"

FILTER_VALIDATE_EMAIL – Verifica la validità di una mail

FILTER_VALIDATE_FLOAT – Verifica che una variabile sia un float valido

FILTER_VALIDATE_INT – Verifica che la variabile sia un intero valido. È possibile anche verificare che sia compreso in un range

FILTER_VALIDATE_IP – Verifica la validità di un IP

FILTER_VALIDATE_URL – Verifica la validità di un URL

Elenco filtri di validazione: https://www.php.net/manual/en/filter.filters.validate.php

CROSS SITE SCRIPTING – XSS (COSA FARE)

SANITIZZARE :

Esempio…

Analizziamo il seguente modulo:

```
<form action=”modulo.php” method=”post”>
        <textarea name=”commento”></textarea>
        <input type=”submit” value=”Invia”>
</form>
```

I dati inseriti nel campo che abbiamo chiamato “commento” vengono inviati alla pagina modulo.php con il seguente codice PHP:

echo $_POST[‘commento’];

Una situazione di questo tipo espone il sito ad un attacco da parte di utenti malintenzionati; Il dato inviato, infatti, non viene filtrato e l’output non è in alcun modo gestito. Se un utente invia il seguente JavaScript: **<script>alert(‘Questa pagina è sotto attacco’);</script>** **ATTACCO DI TIPO CROSS SITE SCRIPTING**

quando il commento viene pubblicato e mostrato sulla pagina, i visitatori che approderanno successivamente su quella pagina vedranno comparire una finestra popup con su scritto “Questa pagina è sotto attacco”.

CROSS SITE SCRIPTING – XSS (COSA FARE)

SANITIZZARE :

Per evitare di incorrere in un problema di questo tipo occorre filtrare e validare tutti i dati che vengono inviati tramite form. Il modo più semplice è il seguente:

$filtra_commento = **strip_tags**($_POST['commento']);

Con la funzione PHP strip_tags() è possibile **"ripulire" (sanitizzare) i dati dei moduli** rimuovendo eventuali tag HTML e PHP inseriti.

Nell'esempio precedente i tag <script> e </script> vengono esclusi dal commento evitando che lo script possa essere eseguito.

Url elenco filtri di sanitizzazione: https://www.php.net/manual/en/filter.filters.sanitize.php

REMOTE CODE INJECTION

La **remote code injection** si riferisce a un'attività in cui un attaccante inserisce e fa eseguire codice malevolo all'interno di un'applicazione web **da una posizione remota**. A differenza dell'XSS, il codice dannoso viene **eseguito dal server** dell'applicazione e non dal browser dell'utente.

L'iniezione remota di codice può coinvolgere vari linguaggi di programmazione, come SQL, JavaScript(NODEJS), PHP o comandi di sistema operativo, a seconda delle tecnologie utilizzate nell'applicazione web.

L'obiettivo principale della Remote Code Injection è sfruttare vulnerabilità nell'applicazione **per assumere il controllo del server o eseguire azioni malevole sul lato server**. Ciò può includere l'accesso a dati sensibili, la manipolazione dell'applicazione o dei sistemi sottostanti, l'esecuzione di comandi di sistema o l'installazione di malware.

XSS E REMOTE CODE INJECTION - CONFRONTO

1. **Cross-Site Scripting (XSS):** L'attacco XSS si verifica quando un attaccante inserisce del codice (solitamente JavaScript) dannoso all'interno delle pagine web visualizzate dagli utenti. Questo codice viene poi **eseguito direttamente dal browser dell'utente**, non dal server. Ciò significa che l'attacco XSS si concentra sull'utente finale e sulla manipolazione dei dati visualizzati nel browser.

2. **Remote Code Injection:** La Remote Code Injection, come spiegato precedentemente, si riferisce a un'attività in cui un attaccante inserisce e fa eseguire codice malevolo all'interno di un'applicazione web da una posizione remota. A differenza dell'XSS, il codice dannoso viene **eseguito dal server** dell'applicazione e non dal browser dell'utente.

SQL INJECTION

- La SQL injection (iniezione di codice SQL) è una vulnerabilità comune nelle applicazioni web che utilizzano un database per archiviare e gestire i dati. Consiste nell'inserimento di codice SQL dannoso all'interno di un'applicazione, sfruttando **input non validati o mal gestiti**, al fine di manipolare le query SQL eseguite dal server del database.
- L'iniezione di codice SQL si verifica quando un attaccante inserisce intenzionalmente del codice SQL dannoso all'interno di un campo di input, come un campo di ricerca o un modulo di accesso, che viene poi utilizzato per creare una query SQL dinamica. Se l'applicazione non gestisce correttamente l'input fornito dagli utenti, il codice SQL dannoso viene incluso nella query e quindi eseguito dal server del database.

SQL INJECTION

L'obiettivo principale della SQL injection è manipolare le query SQL in modo da ottenere informazioni riservate, alterare i dati nel database o eseguire azioni non autorizzate nel sistema. Le conseguenze possono includere l'accesso ai dati sensibili, il compromesso dell'integrità dei dati, il danneggiamento dell'applicazione o addirittura l'assunzione del controllo completo del server.

Ecco un esempio semplificato di SQL injection in un'applicazione di accesso:

Supponiamo che un'applicazione di accesso abbia un campo "nome utente" e un campo "password" che vengono utilizzati per creare una query SQL per verificare le credenziali:

```
SELECT * FROM utenti WHERE username = 'nome_utente' AND password = 'password'
```

SQL INJECTION

Se l'applicazione **non valida o sanifica correttamente l'input** fornito dall'utente,

un attaccante potrebbe inserire un valore malizioso nell'input del campo "nome utente".

Ad esempio, inserendo ' OR '1'='1 come nome utente:

SELECT * FROM utenti WHERE username = '' **OR '1'='1'** AND password = 'password'

Nella query risultante, l'affermazione '1'='1' è sempre vera, quindi l'attaccante potrebbe ottenere l'accesso senza conoscere la password corretta.

SQL INJECTION

Per mitigare la SQL injection, è fondamentale utilizzare **parametri di query o istruzioni preparate**, che consentono ai dati di essere separati dalla struttura della query SQL. Inoltre, l'applicazione dovrebbe eseguire una rigorosa validazione e sanificazione dell'input fornito dagli utenti e limitare i privilegi dell'account del database per ridurre l'impatto potenziale di un'eventuale iniezione di codice SQL.

SQL INJECTION – ESEMPIO QUERY PARAMS

Ecco un esempio di query MySQL utilizzando il **prepared statement**:

Supponiamo che tu abbia un'applicazione web in cui gli utenti possono inserire il loro nome e la loro età, e vuoi eseguire una query per inserire questi dati in una tabella chiamata "utenti".

```
// Connessione al database
$servername = "nome_server";
$username = "nome_utente";
$password = "password";
$dbname = "nome_database";
```

SQL INJECTION – ESEMPIO QUERY PARAMS

```
if (!empty($nome) && !empty($email)) {
    // Connessione al database
    $conn = new mysqli($servername, $username, $password, $dbname);
    if ($conn->connect_error) {
        die("Connessione fallita: " . $conn->connect_error);
    }
    // Query con prepared statement
    $stmt = $conn->prepare("INSERT INTO tabella (nome, email) VALUES (?, ?)");
    $stmt->bind_param("ss", $nome, $email);  // ss sta ad indicare il tipo di dati che ci si aspetta s= stringa, I= int, d= double, b= blob
    // Esecuzione della query
    if ($stmt->execute()) {
        echo "Dati inseriti con successo nel database!";
    } else {
        echo "Errore nell'esecuzione della query: " . $stmt->error;
    }
    // Chiusura dello statement e della connessione
    $stmt->close();
    $conn->close();
}
```

SQL INJECTION – ESEMPIO QUERY PARAMS

La query SQL viene dichiarata con il **prepared statement** utilizzando i **segnaposto** nominati :nome e :email per rappresentare i parametri.

Successivamente, i valori dei parametri nome e email vengono forniti dagli utenti attraverso un form

e vengono associati ai segnaposto nel prepared statement utilizzando il metodo **bindParam().**

Infine, la query viene eseguita tramite il metodo **execute()**

SQL INJECTION – ESEMPIO QUERY PARAMS

ABBIAMO FATTO TUTTO QUANTO POSSIBILE

PER SECURIZZARE LA NOSTRA QUERY?

SQL INJECTION – ESEMPIO QUERY PARAMS

Sicuramente NO !!! Avremmo potuto sicuramente…

1) **Filtrare e validare meglio l'INPUT nome ED email (filter_var, strip_tags / htmlentities etc…)**
2) **Usare PDO invece di MySQli**

SQL INJECTION – PDO VERSUS MYSQLI

MySQLi (MySQL Improved) e PDO (PHP Data Objects) sono due estensioni di PHP che consentono di connettersi e interagire con database MySQL. Tuttavia, PDO è considerato più sicuro rispetto a MySQLi per diversi motivi:

1. **Supporto multipiattaforma**: PDO è un'astrazione dei database che supporta diverse tipologie di database, tra cui MySQL, PostgreSQL, SQLite e altri. Ciò significa che puoi utilizzare lo stesso set di funzioni PDO per interagire con database diversi. Questa caratteristica rende più semplice la migrazione del codice tra database e riduce la probabilità di errori di sicurezza legati alle differenze di implementazione.
2. **Prepared Statement con segnaposto nominati**: PDO supporta i segnaposto nominati nelle query preparate. i segnaposto nominati sono una forma di marcatori di posizione nella creazione di prepared statement. A differenza dei segnaposto tradizionali, che sono rappresentati da punti interrogativi (?), i segnaposto nominati sono identificati da un nome univoco preceduto da due punti (:nome).
3. **Gestione integrata degli errori**: PDO offre una gestione integrata degli errori attraverso l'uso delle eccezioni. Questo semplifica la gestione degli errori e la scrittura di codice più robusto. Inoltre, PDO offre funzionalità come il recupero delle transazioni, il rollback e il commit, che possono essere utili per mantenere l'integrità dei dati e la coerenza nelle operazioni sul database.

SQL INJECTION – PDO VERSUS MYSQLI

```
// prepared statement con Mysqli
   $stmt = $conn->prepare("INSERT INTO tabella (nome, email) VALUES (?, ?)");
   $stmt->bind_param("ss", $nome, $email);
```

```
// prepared statement con PDO
      $stmt = $conn->prepare("INSERT INTO tabella (nome, email) VALUES (:nome, :email)");
      $stmt->bindParam(':nome', $nome);
      $stmt->bindParam(':email', $email);
```

ATTACCHI AL FILE SYSTEM

Gli attacchi al filesystem di solito avvengono anche perché la parte server non è configurata in modo corretto e questi potrebbero portare a problemi particolarmente gravi come ad esempio l'inclusione di file locali per visualizzarne il contenuto (i file a rischio potrebbero essere **/etc/passwd**, i **file di configurazione** o quelli di log), l'alterazione del contenuto delle sessioni (che solitamente sono salvate all'interno della directory /tmp) e la meno comune ma non poco pericolosa alterazione dei file di upload temporanei, oppure a causa di controlli non meticolosi effettuati dagli sviluppatori sui dati utilizzati per l'accesso al filesystem.

ATTACCHI AL FILE SYSTEM

Per proteggersi da questo tipo di attacchi gli sviluppatori dovrebbero assicurarsi che **tutte le variabili utilizzate in questo contesto siano inizializzate prima del loro utilizzo**, assicurarsi che gli utenti **possano influenzare solamente le risorse accessibili per una determinata operazione (ad es. se si prevede il caricamento di immagini, queste dovranno poter essere caricate in una sola cartella del server che NON ABBIA anche i permessi di esecuzione), spostare tutto ciò che non è strettamente legato all'applicazione web al di fuori della document root in modo che non sia accessibile dagli script PHP**.

ATTACCHI AL FILE SYSTEM – CONFIGURAZIONE DI PHP

Anche se non è possibile considerarla come un attacco vero e proprio, la configurazione scorretta di PHP è uno dei maggiori motivi per cui si hanno falle di sicurezza nelle applicazioni web sviluppate con questo linguaggio.

Il grosso problema è che molte opzioni relative alla sicurezza sono impostate in modo non corretto anche nella configurazione standard (quella più adottata dagli hoster) dando un falso senso di sicurezza quando in realtà i problemi possibili sono molti.

ATTACCHI AL FILE SYSTEM – CONFIGURAZIONE DI PHP

- **register_globals**: l'indicazione che debba assolutamente essere impostato ad off è stato ripetuto in tutti i modi, in moltissime situazioni e da moltissime persone, ma pare che non tutti si siano ancora mossi in quella direzione;
- **allow_url_fopen**: questa direttiva è attiva di default ma dovrebbe essere disabilitata per prevenire attacchi di esecuzione remota di codice;
- **magic_quotes_gpc**: dovrebbe essere disabilitata dato che le applicazioni dovrebbero essere progettate e sviluppate affinché si proteggano dalle SQL Injection senza l'ausilio del quoting automatico dell'input;
- **open_basedir**: anche questa direttiva dovrebbe essere abilitata e configurata a dovere per limitare il movimento dell'applicazione solamente all'interno di un ambiente ristretto;

ATTACCHI AL FILE SYSTEM – PROTEZIONE CARTELLE

Per impedire l'accesso a una cartella del tuo web server, puoi utilizzare diverse tecniche a livello di configurazione del server. Ecco alcune opzioni comuni:

File .htaccess: Puoi creare un file .htaccess nella cartella che desideri proteggere e utilizzare le direttive di accesso per impedire l'accesso diretto alla cartella. Ad esempio, puoi utilizzare la direttiva **Deny from all** per negare l'accesso a tutti gli utenti.

Configurazione del server: Se hai accesso diretto alla configurazione del server, puoi impostare regole di accesso specifiche per la cartella che desideri proteggere. Ad esempio, nel caso di Apache, puoi utilizzare la direttiva **<Directory>** nel file di configurazione per definire le regole di accesso.

Indicizzazione disabilitata: Se la cartella contiene file e directory che non devono essere visualizzati pubblicamente, puoi disabilitare l'indicizzazione dei file. In questo modo, se un utente tenta di accedere direttamente alla cartella, non verrà visualizzato un elenco dei file al suo interno.

ATTACCHI AL FILE SYSTEM – PROTEZIONE CARTELLE

Esempio di .htaccess

```
# Impedisci l'accesso diretto alla cartella

Deny from all


# Consenti l'accesso solo da IP specifici

Allow from 123.456.789.123


# Consenti l'accesso solo dall'utente-agent (UserAgentDelTuoSitoWeb$  è una stringa di testo inviata dal client (tipicamente un browser) al server web durante una comunicazione HTTP

RewriteEngine on

RewriteCond %{HTTP_USER_AGENT} !^UserAgentDelTuoSitoWeb$

RewriteRule ^ - [F]
```

ATTACCHI AL FILE SYSTEM – PROTEZIONE CARTELLE

COSA E' LO USER AGENT:

- L'user agent di una richiesta web è una stringa di testo inviata dal client (tipicamente un browser) al server web durante una comunicazione HTTP. L'user agent fornisce informazioni sull'applicazione o sul dispositivo che sta facendo la richiesta al server.
- L'user agent può contenere informazioni come il nome e la versione del browser, il sistema operativo, il dispositivo utilizzato, le caratteristiche supportate e altre informazioni pertinenti. Queste informazioni consentono al server di adattare la risposta in base alle specifiche del client.

ATTACCHI AL FILE SYSTEM – PROTEZIONE CARTELLE

- Esempio di user agent:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.81 Safari/537.36

in questo caso, l'user agent indica che la richiesta proviene da un browser Chrome versione 94 su un sistema operativo Windows 10 a 64 bit.

- L'user agent è spesso utilizzato per scopi di analisi, per determinare la compatibilità del client con determinate funzionalità o per fornire contenuti specifici in base al dispositivo o al browser utilizzato. Tuttavia, è importante notare che l'user agent **può essere manipolato o falsificato**, quindi non può essere considerato una fonte affidabile per determinare con certezza le caratteristiche del client che effettua la richiesta.
- Nella configurazione del server web, **è possibile utilizzare l'user agent per applicare regole specifiche o restrizioni di accesso in base alle specifiche del client.**

PARTE SECONDA

- Securizzare form con l'HTML
- Securizzare form tramite Javascript
- COSA è OWASP

SECURIZZARE UN FORM – CON HTML

- Quando si sviluppa un sito web che include un modulo di invio dati, è fondamentale garantire la sicurezza delle informazioni trasmesse. Gli attacchi informatici possono sfruttare le vulnerabilità dei form HTML per ottenere accesso non autorizzato o compromettere i dati dei visitatori. Per fortuna, esistono diverse misure di sicurezza che è possibile implementare tramite l'uso di tag HTML per proteggere i form e prevenire potenziali attacchi.

Utilizzo del token CSRF / JWT

- Un attacco comune sui form è il **Cross-Site Request Forgery (CSRF),** noto anche come XSRF, è un tipo di attacco informatico che sfrutta **la fiducia tra un utente e un'applicazione web** a cui l'utente ha accesso. In un attacco CSRF, un aggressore induce l'utente a eseguire azioni non intenzionali e non autorizzate su un sito web o un'applicazione in cui l'utente è già autenticato.

SECURIZZARE UN FORM – CON HTML

- Il funzionamento di un attacco CSRF di solito avviene nel seguente modo:

1. **Creazione di una trappola**: L'aggressore crea un sito web o inserisce un codice malevolo all'interno di un sito legittimo, che contiene una richiesta falsificata verso l'applicazione web bersaglio. Questa richiesta falsificata può essere un semplice link, un'immagine o un modulo invisibile.
2. **Induzione dell'utente**: L'utente viene ingannato per visitare il sito compromesso o fare clic su un link malevolo. Questo può essere fatto attraverso l'invio di e-mail ingannevoli, messaggi sui social media o persino tramite siti web compromessi.
3. **Esecuzione dell'azione non autorizzata**: Quando l'utente visita il sito/app compromesso, la richiesta falsificata viene inviata all'applicazione bersaglio utilizzando i cookie o i token di autenticazione validi dell'utente. Poiché l'utente è già autenticato nell'applicazione, l'azione viene eseguita come se fosse stata intenzionalmente avviata dall'utente stesso.

SECURIZZARE UN FORM – CON HTML

- Un esempio pratico di attacco CSRF potrebbe essere il seguente:
- Supponiamo che un **utente autenticato** sia connesso a un social media e stia navigando su un forum di discussione. L'attaccante pubblica un post nel forum contenente un'immagine che, sconosciuta all'utente, contiene un codice HTML con una richiesta falsificata verso il social media.
- Quando l'utente visualizza il post nel forum, il browser carica l'immagine e invia automaticamente la richiesta falsificata al social media, eseguendo un'azione non autorizzata come ad esempio cambiare la password o effettuare una pubblicazione indesiderata.

SECURIZZARE UN FORM – CON HTML

- Per proteggersi dagli attacchi CSRF, le applicazioni web possono adottare diverse misure di sicurezza, tra cui:

1. Utilizzo di token CSRF/JWT: Le applicazioni possono generare token univoci per ogni utente e includerli in ogni richiesta che modifica lo stato dell'applicazione. Questi token vengono quindi verificati dal server per garantire che la richiesta sia stata effettivamente inviata dall'utente autenticato e non da un attaccante.
2. Convalida i dati lato client:
    1. La convalida dei dati lato client ti consente di identificare e correggere gli errori nei dati inseriti dagli utenti **prima che vengano inviati al server**. Puoi utilizzare attributi come "required" e "pattern" per specificare i requisiti dei campi e i formati accettati. Questo aiuterà a prevenire l'invio di dati non validi o dannosi al server.
    2. <input type="email" name="email" required **pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"**>
    3. Esempio uso pattern … https://www.w3schools.com/tags/att_input_pattern.asp

SECURIZZARE UN FORM – CON HTML

Riassunto di cosa si può fare:

Attributo required: Puoi utilizzare l'attributo required su campi di input obbligatori per assicurarti che l'utente compili tutti i campi richiesti prima di inviare il form.

Attributo autocomplete: Puoi utilizzare l'attributo autocomplete per controllare se il browser dovrebbe suggerire o meno i valori precedentemente inseriti nei campi di input. Puoi impostarlo su "off" per evitare che il browser memorizzi e suggerisca i valori dei campi del form.

Attributo pattern: Puoi utilizzare l'attributo pattern per specificare un'espressione regolare che definisce il formato accettato per il valore di un campo di input. Ad esempio, puoi utilizzare un pattern per accettare solo caratteri alfanumerici o un formato specifico per un campo email.

Maxlenght: Se si sa a priori la lunghezza di un campo, limitarlo a quel numero di caratteri

Es. di campo CAP

```
<input type="text" name="codice_postale" pattern="[0-9]{5}" title="Inserisci un codice postale valido a 5 cifre« maxlenght="5" required autocomplete="off">
```

SECURIZZARE UN FORM – CON JAVASCRIPT

Proteggere un form tramite JavaScript implica principalmente la validazione dei dati e la prevenzione di attacchi come Cross-Site Scripting (XSS) e Cross-Site Request Forgery (CSRF).

Validazione dei dati lato client: Puoi utilizzare espressioni regolari o funzioni JavaScript per controllare che i campi siano compilati correttamente e rispettino determinati criteri. Ad esempio, puoi verificare che un campo email sia nel formato corretto.

Prendiamo ad es. il form seguente tipico form di login…

SECURIZZARE UN FORM – CON JAVASCRIPT

```
<form onsubmit="return validateForm()">

  <input type="email" name="email" id="email" required>

  <input type=«text" name="password" id="password" required maxlenght="15">

  <input type=«button" value="Invia">

</form>
```

SECURIZZARE UN FORM – CON JAVASCRIPT – CRIPT PASSWORD

```
<html>
<body>
</body>
<script src="https://unpkg.com/jssha@3.3.0/dist/sha.js"></script>  <!-- CDN LIBRERIA jsSHA -->
<script>
// Funzione per crittografare la password utilizzando l'algoritmo di hash SHA-256
function encryptPassword(password) {
  // Creazione di un oggetto dell'algoritmo SHA-256
  var sha256 = new jsSHA("SHA-256", "TEXT");
  // Calcolo dell'hash della password
  sha256.update(password);
  var encryptedPassword = sha256.getHash("HEX");
  return encryptedPassword;
}
```

SECURIZZARE UN FORM – CON JAVASCRIPT

```
function validateForm(evt) {
evt.preventDefault(); // preveniamo il normale submit del form (anche se il pulsante è di tipo button)
  var emailInput = document.getElementById("email");
  var emailPattern = /^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$/;
  if (!emailPattern.test(emailInput.value)) {
    alert("Inserisci un indirizzo email valido!");
    return false;
  }
var password = document.getElementById(«password");
var encryptedPassword = encryptPassword(password);
…. continua
```

SECURIZZARE UN FORM – CON JAVASCRIPT – LIMIT SUBMIT

```
// Definiamo una variabile per contare il numero di invii
      var submitCount = 0; // da togliere se usato con localStorage
 // Incrementiamo il contatore di invii
 submitCount++;
 // Se il numero di invii supera un certo limite, disabilitiamo il form
 var limit = 3; // Limite di invii consentiti
 if (submitCount >= limit) {
   event.preventDefault(); // Impediamo l'invio del form
   alert("Hai raggiunto il limite di invii consentiti.");
   // Puoi anche nascondere il form o visualizzare un messaggio di errore
 } else { INVIA FORM TRAMITE CHIAMATA AJAX} }
```

SECURIZZARE UN FORM – CON JAVASCRIPT – LIMIT SUBMIT

Per mantenere il numero di tentativi anche se si rilegge la pagina, si potrebbe usare localStorage….

```
// Leggo lo stato del contatore dal localStorage
If (localStorage.getItem('countformSubmitted ') {
var submitCount = localStorage.getItem('countformSubmitted ') } else {var submitCount =  0 }
// Incrementiamo il contatore di invii
  submitCount++;
// Aggiorno nel localStorage che il form è stato inviato
    localStorage.setItem('countformSubmitted', submitCount);
  // Se il numero di invii supera un certo limite, disabilitiamo il form
  var limit = 3; // Limite di invii consentiti
  if (submitCount >= limit) {
    event.preventDefault(); // Impediamo l'invio del form in altri modi
    alert("Hai raggiunto il limite di invii consentiti.");
} else { INVIA FORM TRAMITE CHIAMATA AJAX} }
```

SECURIZZARE UN FORM – CON JAVASCRIPT – LIMIT SUBMIT

RECAP: cosa abbiamo fatto per securizzare un form di LOGIN

1) Impostato i campi a REQUIRED in quanto il form non deve poter essere inviato in mancanza di anche un solo dato
2) Limitata la lunghezza della password a 15 caratteri
3) Validata l'email tramite PATTERN
4) Criptata la password ancor prima di essere inviata al server per evitarne la trasmissione in chiaro
5) Limitato il numero di tentativi a 3 utilizzando anche il localStorage per evitare l'azzeramento dei tentativi tramite reload della pagina

SQL INJECTION – ESEMPIO QUERY PARAMS

ABBIAMO FATTO TUTTO QUANTO POSSIBILE

PER SECURIZZARE IL NOSTRO LOGIN?

SECURIZZARE UN FORM – CON JAVASCRIPT

NO…. MANCA IL CAPTCHA ☺

Definiamo l'HTML che ci serve per visualizzare il CAPTCHA

```
<!-- HTML -->


<div id="captcha">
  <label id="captcha-container"></label>
  <input type="text" id="captcha-input" placeholder="Inserisci i caratteri qui">
  <button onclick="validateCaptcha()">Verifica</button>
</div>
```

SECURIZZARE UN FORM – CON JAVASCRIPT

```
// JavaScript
// Genera un numero casuale per il CAPTCHA
function generateCaptcha() {
  var randomNum = Math.floor(Math.random() * 10000) + 1; // Genera un numero tra 1 e 10000
  return randomNum;
}
// Esempio di utilizzo
var captchaValue = generateCaptcha(); // Genera il valore del CAPTCHA
// Mostra il CAPTCHA all'utente
document.getElementById('captcha-container').innerHTML = captchaValue;
```

SECURIZZARE UN FORM – CON JAVASCRIPT

```
// Controlla se il valore inserito dall'utente corrisponde al CAPTCHA generato
function validateCaptcha() {
var captchaContainer = document.getElementById('captcha-container').innerHTML;
// Ottieni il valore inserito dall'utente
var userInput = document.getElementById('captcha-input').value;
// Convalida il CAPTCHA
if (userInput == captchaValue) {
  // Il CAPTCHA è valido, procedi con l'azione desiderata
  alert('CAPTHA VALIDO');
} else {
  // Il CAPTCHA non è valido, mostra un messaggio di errore o richiedi all'utente di riprovare
  alert('CAPTHA NON VALIDO');
} }
```

COSA E' L'OWSAP

La Open Web Application Security Project (OWASP) è una comunità aperta con lo scopo di permettere alle organizzazioni di sviluppare, vendere e mantenere applicazioni che possono essere considerate sicure.

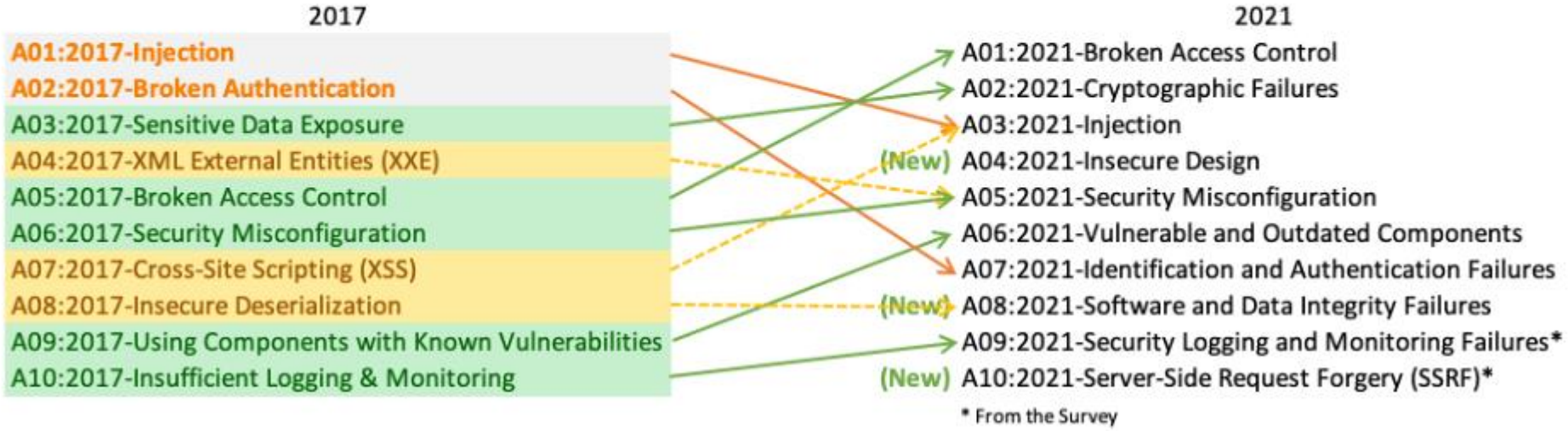
Tutti gli strumenti OWASP, documenti, forums, e capitoli **sono liberi e aperti a chiunque** sia interessato a migliore la sicurezza applicativa. Noi indichiamo l'approccio alla sicurezza applicativa come un problema tecnologico, di processo, della gente perché un approccio effettivo alla sicurezza applicativa include miglioramenti in tutte queste aree.

Potete trovarlo all'indirizzo **https://www.owasp.org**

COSA E' L'OWSAP

Top 10 Web Application Security Risks

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.

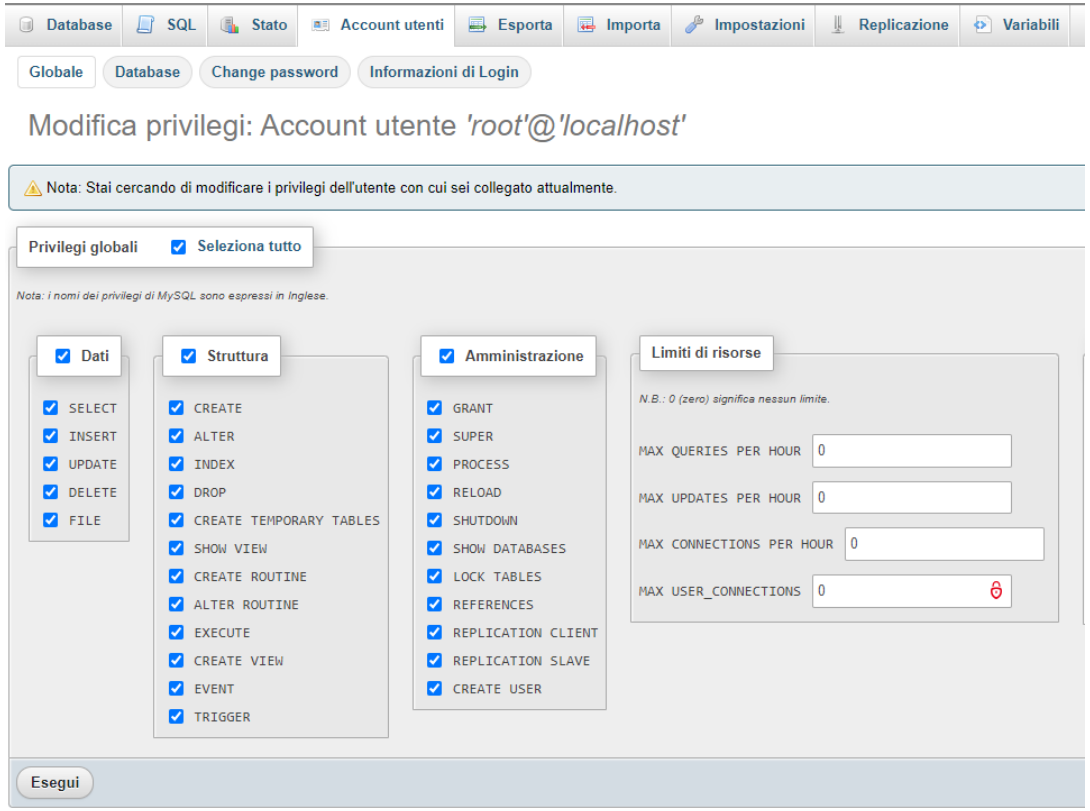


PARTE TERZA

- Prevenire le SQL Injection
- Proteggere l'upload di file
- Proteggere i dati

MYSQL – RICORDIAMOCI DEI PRIVILEGI

Per prima cosa assegniamo ad un utente i privilegi corretti e non usiamo ROOT in produzione ☺ :



Di solito è sufficiente dare i permessi di SELECT, INSERT, UPDATE e DELETE

SQL INJECTION - INTRODUZIONE

Una delle vulnerabilità più comuni è l'iniezione di SQL, che si verifica quando i dati inseriti dagli utenti non vengono adeguatamente controllati o filtrati prima di essere utilizzati nelle query del database. Per mitigare questo rischio, è fondamentale utilizzare i **prepared statement**. Esploreremo ora cosa sono i prepared statement e come utilizzarli con PHP PDO, una delle librerie più popolari per l'accesso ai database in PHP.

Cos'è un prepared statement?

Un prepared statement è un meccanismo offerto dai database che permette di **separare il codice SQL dalle variabili** che verranno utilizzate in esso. Invece di concatenare direttamente i valori nelle query, si creano dei **placeholder**, che verranno successivamente sostituiti con i dati forniti. Questo approccio offre diversi vantaggi in termini di sicurezza e prestazioni.

PREPARED STATEMENT - VANTAGGI

- Vantaggi dei prepared statement:

1. **Protezione contro l'iniezione di SQL**: Utilizzando i prepared statement, si previene l'iniezione di SQL in quanto i valori vengono correttamente trattati come dati e non come parte del codice SQL. I placeholder sostituiscono i valori forniti dall'utente, evitando che i caratteri speciali possano alterare la query.
2. **Prevenzione di errori di sintassi**: I prepared statement gestiscono automaticamente la sintassi del codice SQL, riducendo il rischio di errori di sintassi nelle query.
3. **Miglioramento delle prestazioni**: Utilizzando i prepared statement, il database **può compilare una volta sola la query** e riutilizzarla più volte con dati diversi, migliorando le prestazioni dell'applicazione.

PREPARED STATEMENT – VANTAGGI NELLE PRESTAZIONI

I prepared statement offrono vantaggi in termini di prestazioni principalmente grazie a due fattori: la compilazione della query e la riduzione del numero di richieste al database.

1) **Compilazione della query**: Quando viene utilizzato un prepared statement, la query SQL viene **compilata solo una volta dal database**. Durante la compilazione, il database analizza la struttura della query e ottimizza il suo piano di esecuzione. Questo significa che **l'analisi sintattica e la pianificazione delle operazioni vengono eseguite una sola volta**, riducendo il carico sul database stesso. Di conseguenza, le successive esecuzioni del prepared statement **con dati diversi** possono essere eseguite più rapidamente, poiché la query è già stata compilata e ottimizzata.

PREPARED STATEMENT – VANTAGGI NELLE PRESTAZIONI

2) **Riduzione del numero di richieste al database**: Utilizzando i prepared statement, è possibile eseguire più operazioni utilizzando la stessa query preparata. Ciò comporta una riduzione del numero di richieste inviate al database. Nelle applicazioni web in cui è necessario eseguire molte operazioni di lettura o scrittura sul database, questa riduzione del carico e del traffico di rete (se il DB è su altro server) può comportare un miglioramento significativo delle prestazioni complessive. Inoltre, il risparmio di tempo dovuto alla compilazione della query e alla riduzione delle richieste può diventare evidente in scenari in cui molte query vengono eseguite in rapida successione, come ad esempio cicli o loop.

PREPARED STATEMENT – VANTAGGI NELLE PRESTAZIONI

esempio di codice PHP con PDO in cui viene utilizzata la stessa query preparata per eseguire più operazioni di scrittura // Query preparata per l'inserimento di dati

```
$stmt = $pdo->prepare("INSERT INTO users (name, email) VALUES ( :name, :email)"); // QUERY COMPILATA UNA SOLA VOLTA se fosse messa nel ciclo foreach di cui sotto verrebbe richiamata N volte

// Eseguire più operazioni di inserimento utilizzando la stessa query preparata

$users = [

    ['John Doe', 'john@example.com'],

    ['Jane Smith', 'jane@example.com'],

    ['Mike Johnson', 'mike@example.com']

];

foreach ($users as $user) {

    // Associazione dei valori ai placeholder

    $stmt->bindParam(':name', $user[0]);

    $stmt->bindParam(':email', $user[1]);

    // Esecuzione della query preparata

    $stmt->execute();

}
```

PREPARED STATEMENT – VANTAGGI NELLE PRESTAZIONI

Ripetendo il processo per ogni elemento nell'array **$users**, è possibile eseguire più operazioni di inserimento utilizzando la stessa query preparata, **evitando la necessità di compilare e inviare una nuova query per ogni record**.

Questo approccio riduce il carico sul database e può portare a miglioramenti significativi delle prestazioni, soprattutto quando è necessario eseguire molte operazioni di scrittura consecutive.

PROTEGGERE L'UPLOAD DI FILE - INTRODUZIONE

L'upload di file è una funzionalità comune in molte applicazioni web moderne. Tuttavia, se non gestito correttamente, può creare **vulnerabilità significative nella sicurezza**. È essenziale adottare misure di protezione appropriate per evitare l'inserimento di file dannosi o compromettenti nel sistema.

Cercheremo di vedere le migliori pratiche per proteggere l'upload di file in PHP, riducendo al minimo i rischi di sicurezza e garantendo l'integrità dei dati.

PROTEGGERE L'UPLOAD DI FILE -

1) Validazione del file: La prima linea di difesa per proteggere l'upload di file consiste nella validazione accurata dei file caricati.
    a) **Verifica dell'estensione del file**: Controlla che l'estensione del file corrisponda a quelle consentite per il tipo di file che si desidera consentire. (si può fare **anche** lato client)
    b) **Controllo del tipo di file**: Utilizza la funzione **finfo_file()** o **mime_content_type()** vedi https://www.php.net/manual/en/function.mime-content-type.php per verificare che il tipo MIME del file sia conforme alle aspettative.

PROTEGGERE L'UPLOAD DI FILE -

- Il tipo MIME (Multipurpose Internet Mail Extensions) di un file è una designazione standardizzata che indica la natura e il formato di un file su Internet. Il tipo MIME **è una stringa di testo che viene associata a un file** per identificarne il contenuto in modo che i browser e le applicazioni possano interpretarlo correttamente.
- Il tipo MIME è composto da **due parti principali**: **il tipo multimediale e il sottotipo**. Il tipo multimediale rappresenta la categoria generale del contenuto, come testo, immagine, audio, video, etc. Il sottotipo specifica il formato specifico all'interno di quella categoria, ad esempio, per i file di immagini potrebbe essere JPEG, PNG o GIF.
- Un tipo MIME viene solitamente **rappresentato come una stringa di testo** nel formato "tipo/sottotipo". Ad esempio, il tipo MIME per un file JPEG potrebbe essere "image/jpeg", mentre per un file PDF potrebbe essere "application/pdf".
- I tipi MIME sono importanti perché consentono ai server web di **inviare il giusto tipo di contenuto al browser o all'applicazione che lo richiede**. Inoltre, i tipi MIME vengono utilizzati per associare le estensioni dei file con il tipo corrispondente, ad esempio, associando l'estensione ".jpg" al tipo MIME "image/jpeg".

PROTEGGERE L'UPLOAD DI FILE -

1) Limiti di dimensione del file: Imposta limiti di dimensione massima per evitare il caricamento di file eccessivamente grandi che potrebbero sovraccaricare il sistema.

2) **Rinominazione dei file**: Una pratica comune per prevenire il sovrascrittura indesiderata o l'accesso non autorizzato consiste nel rinominare i file durante il processo di upload. Utilizza una logica di rinominazione unica, ad esempio combinando un **timestamp** e un **identificatore univoco (https://www.php.net/manual/en/function.uniqid.php)**, per evitare collisioni di nomi di file e preservare la privacy degli utenti.

3) **Posizione di archiviazione sicura**: Assicurati che i file caricati siano archiviati in una **directory non accessibile direttamente tramite il web server**. Allo stesso tempo, garantisci che il percorso di archiviazione abbia **le autorizzazioni corrette** in modo che il server possa leggere e scrivere i file quando necessario. Ciò impedisce agli utenti di accedere direttamente ai file caricati, aumentando la sicurezza del sistema.

PROTEGGERE L'UPLOAD DI FILE -

4) **Sanitizzazione dei nomi di file**:

Prima di salvare il nome di file nel database o utilizzarlo per scopi di visualizzazione, assicurati di sanitarizzarlo per evitare attacchi di cross-site scripting (XSS). Utilizza la funzione **htmlspecialchars()** per convertire i caratteri speciali in entità HTML.

5) **Validazione lato server**: Anche se la validazione lato client (JavaScript) è importante per fornire feedback immediato agli utenti, la validazione lato server è fondamentale per garantire l'integrità dei dati. Ripeti la validazione del file anche nel backend prima di elaborare l'upload effettivo. Questo impedirà il caricamento di file dannosi o non conformi tramite bypass della validazione lato client.

PROTEGGERE L'UPLOAD DI FILE -

6) **Limitare i permessi di esecuzione**: Imposta i permessi di file adeguati per i file caricati, limitando l'esecuzione dei file caricati laddove necessario. Ad esempio, per i file immagine, potresti limitare l'esecuzione solo alla visualizzazione delle immagini e impedire l'esecuzione di script incorporati.

PROTEGGERE I DATI - INTRODUZIONE

La protezione dei dati è fondamentale in qualsiasi applicazione che utilizza un database per archiviare informazioni sensibili. Una violazione dei dati può causare danni significativi, compromettere la fiducia degli utenti e avere conseguenze legali. Esploreremo le migliori pratiche per proteggere i dati nel tuo database, fornendo linee guida per una gestione sicura e affidabile.

PROTEGGERE I DATI -

1. **Utilizza accessi e autorizzazioni appropriati**: Una delle prime misure di sicurezza da adottare è garantire che solo le persone autorizzate abbiano accesso al database.
2. **Imposta password robuste** per gli account degli utenti e non utilizzare mai le password di default.
3. Limita l'accesso dei singoli utenti solo alle risorse di database necessarie per svolgere le proprie funzioni.
4. **Evita di utilizzare account amministrativi** per scopi non amministrativi.
5. Monitora e registra gli accessi per individuare attività sospette.

PROTEGGERE I DATI -

6) **Cripta i dati sensibili**: La crittografia dei dati sensibili protegge le informazioni confidenziali nel database. Considera l'utilizzo della crittografia a livello di campo o a livello di disco per proteggere i dati in transito e a riposo. Ad esempio, puoi utilizzare algoritmi di crittografia come AES (Advanced Encryption Standard) per proteggere le informazioni sensibili come numeri di carta di credito o password.

7) **Esegui regolari backup dei dati**: I backup regolari sono essenziali per garantire la sicurezza dei dati in caso di eventi catastrofici, errori umani o attacchi informatici. Assicurati di pianificare e automatizzare i backup del tuo database, inclusi i file di registro delle transazioni. Verifica anche la validità dei backup eseguendo test di ripristino periodici.

8) **Sanitizza e valida i dati in ingresso**

9) **Proteggi dalle vulnerabilità di iniezione di SQL**

RIASSUMIAMO PER PUNTI QUANTO VA FATTO

RECAP …. RICORDIAMOCI QUESTI PUNTI - 1/2

Pratiche che un programmatore dovrebbe adottare per scrivere codice sicuro in PHP con MySQL:

- **Utilizzare prepared statement o istruzioni preparate**: Utilizzare sempre prepared statement o istruzioni preparate per eseguire query al database. Questo aiuta a separare i dati dall'SQL e previene l'SQL injection. I parametri vengono associati in modo sicuro alla query, evitando la concatenazione diretta di valori.
- **Validare l'input degli utenti**: Prima di utilizzare l'input fornito dagli utenti, è essenziale eseguire una valida e approfondita validazione. Assicurarsi che i dati inseriti rispettino i criteri previsti, come lunghezza, formato e tipo. Inoltre, utilizzare funzioni di escape o filtri appropriati per sanificare l'input e rimuovere caratteri dannosi o non validi. **Da fare sia lato CLIENT che lato SERVER**
- **Limitare i privilegi dell'account del database**: Assicurarsi che l'account utilizzato per connettersi al database abbia solo i privilegi necessari per eseguire le operazioni richieste. Evitare di utilizzare un account con privilegi di amministratore per le operazioni di routine dell'applicazione.

RECAP …. RICORDIAMOCI QUESTI PUNTI - 2/2

- **Gestire gli errori in modo sicuro**: Evitare di mostrare informazioni dettagliate sugli errori al pubblico. Impostare l'opzione **display_errors su Off in produzione** per evitare che le informazioni sensibili vengano visualizzate agli utenti. Invece, registrare gli errori in un file di log o in un sistema di monitoraggio per una corretta analisi e risoluzione dei problemi.
- **Aggiornare e patchare il software**: Assicurarsi di utilizzare le versioni più recenti di PHP, MySQL e delle librerie di connessione al database. Mantenere il software aggiornato e applicare regolarmente le patch di sicurezza rilasciate dagli sviluppatori.
- **Utilizzare HTTPS per la comunicazione sicura**: Utilizzare il protocollo HTTPS per crittografare la comunicazione tra il server web e il browser dell'utente. Ciò protegge i dati sensibili durante il trasferimento e previene attacchi come l'intercettazione dei dati.

RECAP …. RICORDIAMOCI QUESTI PUNTI - 2/2

- **Proteggere l'upload dei files**
- **Fare backup regolari e testarne il funzionamento**
- **Criptare i dati sensibili nei campi del DB**
- **Non rendere accedibile la cartella degli Upload direttamente dal Web**
- **Dare i giusti permessi alla cartella degli Upload**

Ricorda… che la sicurezza è un **processo continuo** e richiede una combinazione di pratiche di sviluppo sicuro, configurazione corretta del server e aggiornamenti regolari per mantenere l'applicazione protetta.

GRAZIE….

GIANPIERO FASULO