Come Creare uno Script Sicuro per il Login Usando PHP e MySQL

Dopo aver letto numerose storie di violazioni di siti web e database, gli sviluppatori stanno continuamente cercando il modo migliore per rendere sicure le proprie informazioni nel web. Se il tuo sito comprende funzionalità per la registrazione e la gestione di utenti, potrebbe essere esposto al rischio di essere violato, con la conseguente perdita di informazioni preziose riguardanti i tuoi utenti. Questa guida ti mostra il modo migliore per creare un sistema di login e di gestione degli utenti che sia il più sicuro possibile.

Segui le istruzioni presenti in questa guida per prevenire ogni tipo di attacco informatico il cui scopo sia guadagnare l'accesso ad altri profili utente, cancellare account o modificare dei dati. Ecco una lista dei possibili attacchi informatici da cui i passaggi di questa guida ti aiuteranno a difenderti.

- SQL Injections
- Session Hijacking
- Network Eavesdropping
- Cross Site Scripting
- Brute Force Attacks

Metodo 2 di 8: Configura un Database mySQL

1. 1

   Crea il database MySQL.
   In questa guida verrà creato un database chiamato 'secure_login'.
   Utilizza il codice SQL che trovi di seguito per creare il tuo database.

   Codice SQL per la creazione di un database:

   ```
   CREATE DATABASE `secure_login`;
   ```

   Nota: alcuni servizi di hosting non ti permettono di creare un database usando l'interfaccia phpMyAdmin, quindi impara a creare un database mySQL usando cPanel.

2. 2

   Crea un utente che abbia solo i privilegi per eseguire 'SELECT', 'UPDATE' e 'INSERT'.
   In questo modo, se venisse scoperta una breccia nella sicurezza dei tuoi script, un hacker non potrebbe cancellare niente dalla struttura o dai dati presenti nel tuo database. Usando un utente di questo tipo sarai in grado di fare praticamente tutto quello che desideri con la tua applicazione. Se sei particolarmente ansioso riguardo alla sicurezza del tuo database, crea un utente diverso per ogni funzione.

   - **Utente:** "sec_user"
   - **Password:** "eKcGZr59zAa2BEWU"

   Codice per la creazione dell'utente 'sec_user':

   ```
   CREATE USER 'sec_user'@'localhost' IDENTIFIED BY 'eKcGZr59zAa2BEWU';
   GRANT SELECT, INSERT, UPDATE ON `secure_login`.* TO
   'sec_user'@'localhost';
   ```

   Nota: è un'ottima idea personalizzare la password del codice descritto qui sopra quando lo esegui sul tuo server. Assicurati anche di cambiare il codice PHP. Ricorda che non si tratta di una password che dovrai ricordare facilmente quindi scegline una il più possibile complicata. A questo indirizzo troverai un generatore di password casuali.

3. 3

   Crea una tabella mySQL chiamata 'members'.
   Il codice che segue creerà una tabella composta da 5 campi (id, username, email, password, salt).

   Codice per la creazione della tabella 'members':

   ```
   CREATE TABLE `secure_login`.`members` (
   ```

```
  `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `username` VARCHAR(30) NOT NULL,
  `email` VARCHAR(50) NOT NULL,
  `password` CHAR(128) NOT NULL,
  `salt` CHAR(128) NOT NULL
) ENGINE = InnoDB;
```

Abbiamo usato il tipo dato 'CHAR' per i campi di cui la dimensione è nota. Per i campi 'password' e 'salt' è stata usata una dimensione massima pari a 128 caratteri. Usando il tipo di dato 'CHAR', avrai un risparmio in termini di utilizzo della CPU.

4. 4

   Crea la tabella di log che memorizzerà tutti i tentativi di login.
   Abbiamo creato questa tabella per memorizzare tutti i tentativi di login da parte degli utenti, per poter prevenire un attacco basato sulla 'forza bruta'.

   Codice per la creazione della tabella 'login_attempts':

```
CREATE TABLE `secure_login`.`login_attempts` (
  `user_id` INT(11) NOT NULL,
  `time` VARCHAR(30) NOT NULL
) ENGINE=InnoDB
```

5. 5

   Esegui un test di inserimento dati nella tabella 'members'.
   Dato che non abbiamo creato una pagina per la registrazione di nuovi utenti, sarà fondamentale testare la robustezza e l'efficienza dello script di login. Di seguito troverai lo script SQL per creare un utente con i seguenti dettagli:

   - **Nome utente:** test_user
   - **Indirizzo E-mail:** test@example.com
   - **Password:** 6ZaxN2Vzm9NUJT2y

   Aggiungi l'utente 'test user':

```
INSERT INTO `secure_login`.`members` VALUES(1, 'test_user',
'test@example.com',
'00807432eae173f652f2064bdca1b61b290b52d40e429a7d295d76a71084aa96c0233b82f
1feac45529e0726559645acaed6f3ae58a286b9f075916ebf66cacc',
'f9aab579fc1b41ed0c44fe4ecdbfcdb4cb99b9023abb241a6db833288f4eea3c02f76e0d3
5204a8695077dcf81932aa59006423976224be0390395bae152d4ef');
```

Metodo 3 di 8: Crea la Pagina per la Connessione al Database

1. 1

   Crea la pagina per la connessione al database.
   Questo è il codice php che noi utilizzeremo per la connessione al nostro database mySQL. Crea un nuovo file php chiamato 'db_connect.php' ed inserisci al suo interno il codice che segue. Potrai inserire questo file in tutte le pagine che necessitano di collegarsi al database.

   Codice php per la connessione al database ('db_connect.php'):

```
define("HOST", "localhost"); // E' il server a cui ti vuoi connettere.
define("USER", "sec_user"); // E' l'utente con cui ti collegherai al DB.
define("PASSWORD", "eKcGZr59zAa2BEWU"); // Password di accesso al DB.
define("DATABASE", "secure_login"); // Nome del database.
$mysqli = new mysqli(HOST, USER, PASSWORD, DATABASE);
// Se ti stai connettendo usando il protocollo TCP/IP, invece di usare un
socket UNIX, ricordati di aggiungere il parametro corrispondente al numero
di porta.
```


Metodo 4 di 8: Crea le Funzioni PHP

Queste funzioni saranno alla base di tutte le operazioni da compiere per eseguire correttamente lo script di login. Aggiungi tutte queste funzioni ad una pagina chiamata 'functions.php'.

1. 1

   Avvia una sessione PHP protetta.
   È importante non usare solo la funzione 'session_start();' all'inizio di ogni pagina che userai all'interno della sessione PHP. Se hai veramente a cuore la sicurezza del tuo sistema, ecco come dovresti realizzare questa sezione del codice. Andremo a creare una funzione chiamata 'sec_session_start' che creerà una sessione php protetta. Dovrai richiamare questa funzione all'inizio di ogni pagina che abbia la necessità di accedere ad una variabile di sessione. Se sei molto preoccupato della sicurezza e della privacy dei dati memorizzati nei tuoi cookie, approfondisci l'argomento eseguendo una ricerca nel web relativa alla gestione di una sessione protetta con un sistema basato su PHP e mySQL.

   Codice relativo alla funzione per la creazione di una sessione php sicura:

```
function sec_session_start() {
        $session_name = 'sec_session_id'; // Imposta un nome di sessione
        $secure = false; // Imposta il parametro a true se vuoi usare il
protocollo 'https'.
        $httponly = true; // Questo impedirà ad un javascript di essere in
grado di accedere all'id di sessione.
        ini_set('session.use_only_cookies', 1); // Forza la sessione ad
utilizzare solo i cookie.
        $cookieParams = session_get_cookie_params(); // Legge i parametri
correnti relativi ai cookie.
        session_set_cookie_params($cookieParams["lifetime"],
$cookieParams["path"], $cookieParams["domain"], $secure, $httponly);
        session_name($session_name); // Imposta il nome di sessione con
quello prescelto all'inizio della funzione.
        session_start(); // Avvia la sessione php.
        session_regenerate_id(); // Rigenera la sessione e cancella quella
creata in precedenza.
}
```

   Questa funzione rende il tuo script di login molto più sicuro. Impedisce ad un hacker di accedere all'id della tua sessione php usando un semplice script java (come avviene ad esempio nel caso di un attacco XSS).
   Inoltre, usando la funzione 'session_regenerate_id()' che rigenera l'id di sessione ad ogni caricamento di pagina, preverrai l'hijacking della sessione.

   Nota: se stai usando il protocollo 'https' per la tua applicazione di login, ricordati di impostare la variabile '$secure' a 'true'.

2. 2

   Crea la funzione di login.
   Questa funzione si occuperà di verificare l'indirizzo e-mail e la password confrontandole con quelle memorizzate nel database, restituendo il valore 'true' nel caso venga trovata una corrispondenza.

Secure Login Function:

```
function login($email, $password, $mysqli) {
   // Usando statement sql 'prepared' non sarà possibile attuare un
attacco di tipo SQL injection.
   if ($stmt = $mysqli->prepare("SELECT id, username, password, salt FROM
members WHERE email = ? LIMIT 1")) {
      $stmt->bind_param('s', $email); // esegue il bind del parametro
'$email'.
      $stmt->execute(); // esegue la query appena creata.
      $stmt->store_result();
      $stmt->bind_result($user_id, $username, $db_password, $salt); //
recupera il risultato della query e lo memorizza nelle relative variabili.
      $stmt->fetch();
      $password = hash('sha512', $password.$salt); // codifica la password
usando una chiave univoca.
      if($stmt->num_rows == 1) { // se l'utente esiste
         // verifichiamo che non sia disabilitato in seguito
all'esecuzione di troppi tentativi di accesso errati.
         if(checkbrute($user_id, $mysqli) == true) {
            // Account disabilitato
            // Invia un e-mail all'utente avvisandolo che il suo account è
stato disabilitato.
            return false;
         } else {
         if($db_password == $password) { // Verifica che la password
memorizzata nel database corrisponda alla password fornita dall'utente.
            // Password corretta!
               $user_browser = $_SERVER['HTTP_USER_AGENT']; // Recupero il
parametro 'user-agent' relativo all'utente corrente.

               $user_id = preg_replace("/[^0-9]+/", "", $user_id); // ci
proteggiamo da un attacco XSS
               $_SESSION['user_id'] = $user_id;
               $username = preg_replace("/[^a-zA-Z0-9_\-]+/", "",
$username); // ci proteggiamo da un attacco XSS
               $_SESSION['username'] = $username;
               $_SESSION['login_string'] = hash('sha512',
$password.$user_browser);
               // Login eseguito con successo.
               return true;
         } else {
            // Password incorretta.
            // Registriamo il tentativo fallito nel database.
            $now = time();
            $mysqli->query("INSERT INTO login_attempts (user_id, time)
VALUES ('$user_id', '$now')");
            return false;
         }
      }
      } else {
         // L'utente inserito non esiste.
         return false;
      }
   }
}
```

3. 3

Funzione per prevenire un attacco che utilizza la "forza bruta".
Gli attacchi informatici basati sulla forza bruta si verificano quando un hacker esegue migliaia di tentativi di accesso ad un sistema usando un account specifico, generando password casuali o prelevandole da un dizionario. Nel nostro script, se un utente esegue 5 tentativi di login errati consecutivi, il suo account viene disabilitato.

Crea la funzione 'checkbrute':

```
function checkbrute($user_id, $mysqli) {
   // Recupero il timestamp
   $now = time();
   // Vengono analizzati tutti i tentativi di login a partire dalle ultime
due ore.
   $valid_attempts = $now - (2 * 60 * 60);
   if ($stmt = $mysqli->prepare("SELECT time FROM login_attempts WHERE
user_id = ? AND time > '$valid_attempts'")) {
      $stmt->bind_param('i', $user_id);
      // Eseguo la query creata.
      $stmt->execute();
      $stmt->store_result();
      // Verifico l'esistenza di più di 5 tentativi di login falliti.
      if($stmt->num_rows > 5) {
         return true;
      } else {
         return false;
      }
   }
}
```

Gli attacchi basati sulla forza bruta sono difficili da prevenire. Alcuni modi per farlo consistono nell'usare i codici CAPTCHA, nel disabilitare l'utente e nell'introdurre un ritardo nell'esecuzione del prossimo tentativo di login dopo un tentativo errato. Questo significa che l'utente dovrà attendere un certo intervallo di tempo (ad esempio 30 secondi) prima di poter effettuare nuovamente il login.
Molti sviluppatori, quando si trovano di fronte a problemi simili, bloccano semplicemente l'accesso al sistema da quel determinato indirizzo IP dopo un certo numero di tentativi di login errati. Alcuni strumenti usati per automatizzare questo tipo di attacco utilizzano server proxy e cambiano indirizzo IP ad ogni richiesta di login. Questo significa che bloccare tutti questi indirizzi IP potrebbe anche impedire il login ad utenti legittimi.

4. 4

Verifica lo stato del login.
Per farlo dobbiamo controllare le variabili di sessione 'user_id' e 'login_string'. La variabile di sessione 'login_string' memorizza le informazioni relative al browser dell'utente codificate insieme alla password di accesso. Utilizziamo le informazioni del browser perché è veramente poco probabile che l'utente lo cambi durante la sessione. Questa accortezza ci aiuta a prevenire un attacco relativo all'hijacking della sessione.

Crea la funzione 'login_check':

```
function login_check($mysqli) {
```

```
   // Verifica che tutte le variabili di sessione siano impostate
correttamente
   if(isset($_SESSION['user_id'], $_SESSION['username'],
$ SESSION['login string'])) {
     $user_id = $_SESSION['user_id'];
     $login_string = $_SESSION['login_string'];
     $username = $_SESSION['username'];
     $user_browser = $_SERVER['HTTP_USER_AGENT']; // reperisce la stringa
'user-agent' dell'utente.
     if ($stmt = $mysqli->prepare("SELECT password FROM members WHERE id =
? LIMIT 1")) {
        $stmt->bind_param('i', $user_id); // esegue il bind del parametro
'$user_id'.
        $stmt->execute(); // Esegue la query creata.
        $stmt->store_result();

        if($stmt->num_rows == 1) { // se l'utente esiste
           $stmt->bind_result($password); // recupera le variabili dal
risultato ottenuto.
           $stmt->fetch();
           $login_check = hash('sha512', $password.$user_browser);
           if($login_check == $login_string) {
              // Login eseguito!!!!
              return true;
           } else {
              //  Login non eseguito
              return false;
           }
        } else {
            // Login non eseguito
            return false;
        }
     } else {
        // Login non eseguito
        return false;
     }
   } else {
     // Login non eseguito
     return false;
   }
}
```

Metodo 5 di 8: Crea le Pagine di Gestione del Processo

1. 1

   Crea la pagina che eseguirà il login (process_login.php).
   Se hai seguito i nostri esempi precedenti, la pagina si chiamerà 'process_login.php'.
   Noi useremo le funzioni PHP incluse nell'estensione mysqli_* di mySQL, perché risulta essere la più aggiornata.

   Crea il codice della pagina di login (process_login.php)

```
include 'db_connect.php';
include 'functions.php';
sec session start(); // usiamo la nostra funzione per avviare una sessione
php sicura
if(isset($_POST['email'], $_POST['p'])) {
   $email = $_POST['email'];
   $password = $_POST['p']; // Recupero la password criptata.
   if(login($email, $password, $mysqli) == true) {
      // Login eseguito
      echo 'Success: You have been logged in!';
   } else {
      // Login fallito
      header('Location: ./login.php?error=1');
   }
} else {
   // Le variabili corrette non sono state inviate a questa pagina dal
metodo POST.
   echo 'Invalid Request';
}
```

2. 2

   Crea lo script di logout.
   Il tuo script di logout deve avviare una sessione, distruggerla e poi reindirizzarti ad un'altra pagina.

   Crea il codice per lo script di logout (logout.php):

```
include 'functions.php';
sec_session_start();
// Elimina tutti i valori della sessione.
$_SESSION = array();
// Recupera i parametri di sessione.
$params = session_get_cookie_params();
// Cancella i cookie attuali.
setcookie(session_name(), '', time() - 42000, $params["path"],
$params["domain"], $params["secure"], $params["httponly"]);
// Cancella la sessione.
session_destroy();
header('Location: ./');
```

   Nota: a questo punto potrebbe essere una buona idea aggiungere la protezione 'CSRF'. Così

facendo, se qualcuno riuscisse in qualche modo ad arrivare all'indirizzo di questa pagina nascosta, l'utente verrebbe disconnesso.

3. 3

Pagina di Registrazione.
Per criptare la password avremo la necessità di usare il seguente codice:

Script per l'hashing della password:

```
// Recupero la password criptata dal form di inserimento.
$password = $_POST['p'];
// Crea una chiave casuale
$random_salt = hash('sha512', uniqid(mt_rand(1, mt_getrandmax()), true));
// Crea una password usando la chiave appena creata.
$password = hash('sha512', $password.$random_salt);
// Inserisci a questo punto il codice SQL per eseguire la INSERT nel tuo
database
// Assicurati di usare statement SQL 'prepared'.
if ($insert_stmt = $mysqli->prepare("INSERT INTO members (username, email,
password, salt) VALUES (?, ?, ?, ?)")) {
   $insert_stmt->bind_param('ssss', $username, $email, $password,
$random_salt);
   // Esegui la query ottenuta.
   $insert_stmt->execute();
}
```

Assicurati che il valore del parametro '$_POST['p']' sia già stato criptato utilizzando un javascript. Se non userai questo metodo, perché vuoi validare la password direttamente sul server, assicurati che sia criptata.

Metodo 6 di 8: Crea i File Javascript

1. 1

   Crea il file 'sha512.js'.
   Questo file è un'implementazione dell'algoritmo di codifica 'sha512' per i javascript. Useremo la funzione di codifica in modo che le nostre password non possano essere inviate in chiaro.

   Il file può essere scaricato dal sito pajhome.org.uk

2. 2

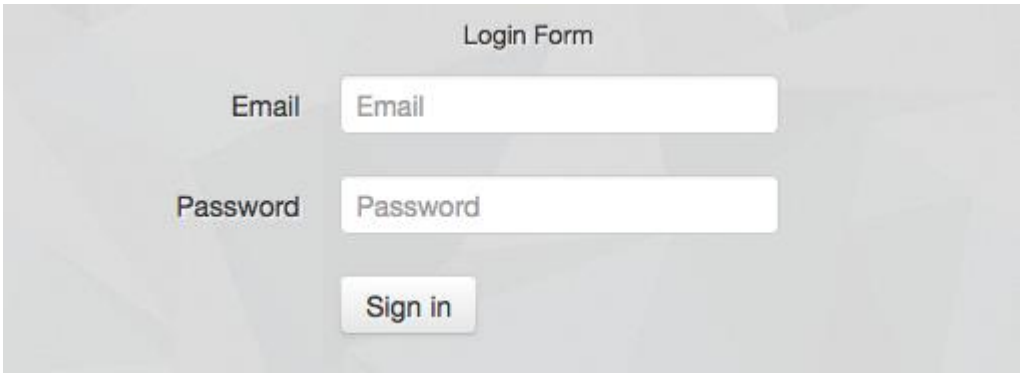
   Crea il file 'forms.js'.
   Questo file provvederà alla codifica delle password per tutti i form.

   Codice per il javascript del file 'forms.js':

```
function formhash(form, password) {
   // Crea un elemento di input che verrà usato come campo di output per
la password criptata.
   var p = document.createElement("input");
   // Aggiungi un nuovo elemento al tuo form.
   form.appendChild(p);
   p.name = "p";
   p.type = "hidden"
   p.value = hex_sha512(password.value);
   // Assicurati che la password non venga inviata in chiaro.
   password.value = "";
   // Come ultimo passaggio, esegui il 'submit' del form.
   form.submit();
}
```

   Pubblicità

Metodo 7 di 8: Crea le Pagine HTML

1. 

   1

   Crea il modulo di login (login.php).

   Sarà composto da un form HTML che conterrà due campi di input chiamati 'e-mail' e

'password'. Lo script java si occuperà della codifica della password e dell'invio al server dei parametri 'email' e 'p' (la password criptata).

Per la fase di login è meglio utilizzare informazioni che non siano pubbliche. In questa guida utilizziamo l'indirizzo e-mail nella fase di login, mentre il nome utente può essere usato per identificare l'utente. Se l'indirizzo e-mail è nascosto, aggiungerai una variabile in più, che dovrà essere gestita da un eventuale hacker.

Codice per la creazione del modulo di login in HTML:

```
<script type="text/javascript" src="sha512.js"></script>
<script type="text/javascript" src="forms.js"></script>
<?php
if(isset($_GET['error'])) {
   echo 'Error Logging In!';
}
?>
<form action="process_login.php" method="post" name="login_form">
   Email: <input type="text" name="email" /><br />
   Password: <input type="password" name="p" id="password"/><br />
   <input type="button" value="Login" onclick="formhash(this.form,
this.form.password);" />
</form>
```

Nota: anche se abbiamo criptato la password in modo che non venga inviata in chiaro, è raccomandato utilizzare il protocollo 'https' (TLS/SSL) per inviare la password al server. (Consulta questa documentazione riguardante il 'Network Eavesdropping').

Metodo 8 di 8: Protezione delle Pagine

1. 1

   Script per la protezione delle pagine.
   Uno dei problemi più comuni rilevabili nei sistemi di autenticazione è che gli sviluppatori si dimenticano di verificare che l'utente sia effettivamente 'loggato' nel sistema. È molto importante che tu faccia uso del seguente codice per verificare che l'utente abbia eseguito il login con successo.
   Codice per la protezione dell'accesso alle pagine:

```
// Inserisci in questo punto il codice per la connessione al DB e
l'utilizzo delle varie funzioni.
sec session start();
if(login_check($mysqli) == true) {

   // Inserisci qui il contenuto delle tue pagine!

} else {
   echo 'You are not authorized to access this page, please login. <br/>';
}
```