Web App Password Manager

Alessandro Bottazzi & Luca Oppici

Ottobre 2024

1 Abstract

L'obiettivo di questo progetto è la realizzazione di una web-app per la gestione e la protezione delle password. Ricordarsi le password, o più in generale dati sensibili, è uno dei problemi che una persona si trova ad affrontare almeno una nella vita; spesso si ricorre a metodi inadeguati, come l'annotazione su carta o la creazione di file di testo appositi. Questo progetto offre una soluzione più efficace, dove, attenzione particolare è stata rivolta anche alla sicurezza contro attacchi informatici, come: SQL injection, dictionary attack, rainbowtable attack, etc. L'applicativo integra misure per prevenire tali vulnerabilità, garantendo una protezione robusta non solo per le password, ma anche per i dati associati agli utenti.

2 Requisiti

In questa sezione andremo ad affrontare il tema dei requisiti, i requisiti sono divisi in funzione delle varie entità quali: user, admin (o operatore tecnico), sistema.
Nella sezione successiva verranno ripresi i requisiti e verranno esposte le relative scelte atte al soddisfacimento di quest'ultimi.

2.1 Requisiti di sistema

I requisiti di sistema riguardano prevalentemente la sicurezza del sistema

- I dati rilevanti devono essere salvati su un database
- Corretta gestione della sessione utente
- La password dell'account utente non devono essere salvate in chiaro
- Le password inserite dagli utenti non devono essere salvare in chiaro ma devono essere criptate con una `masterPassword`. Quest'ultima viene chiesta una prima volta durante la registrazione ed è compito dell'utente non dimenticarla altrimenti tutti i dati cifrati andranno persi.

2.2 Requisiti user

I requisiti dello user riguardano le funzionalità del programma relative alle azioni intraprendibili e all'interfaccia lato client.

- L'utente dev'essere in grado di creare un proprio account
- L'utente dev'essere in grado di creare, rimuovere e vedere le password che ha salvato e le loro relative informazioni
- L'utente deve avere la possibilità di cambiare la master password
- L'utente dev'essere in grado di uscire dal proprio account tramite un bottone log-out

2.3 Requisiti admin

- Visione dei possibili attacchi di tipo SQL Injection con relative informazioni

3 Progettazione

La fase di progettazione è la fase più delicata dell'intero progetto in quanto il grado di sicurezza è strettamente legato a quelle che sono le tecniche di difesa adottate. In questa sezione parleremo degli aspetti tecnici e delle scelte che sono state prese affinchè i vari requisiti venissero soddisfatti.

3.1 Bcrypt e `usernamePassword`

Con `usernamePassword` ci si riferisce al campo password della tupla username-password quando si effettua la registrazione sul sito web, sono quindi le credenziali. Questa password verrà salvata sotto forma di hash. L'algoritmo di hash usato è: bcrypt. E' un algoritmo che utilizza un salt per computare l'hash ed è, volutamente, computazionalmente costoso; è progettato per essere sufficientemente lento per evitare attacchi a forza bruta. Inoltre è un algoritmo con elevata scalabilità in quanto è possibile andare a modificare il parametro `COST_FACTOR` che determina il numero di round di hash. Un prototipo di utilizzo è:

```
hash_bcrypt = bcrypt(usernamePassword, COST_FACTOR)
```

Dove il risultato è una stringa con il seguente formato:

```
$2a$12$R9h/cIPz0gi.URNNX3kh2OPST9/PgBkqquzi.Ss7KIUgO2t0jWMUW
\__/\/ \____________________/\_____________________________/
Alg Cost      Salt                        Hash
```

3.2 Cifratura a chiave simmetrica e `masterPassword`

Le password che l'utente desidera salvare nel proprio account, per ovvi motivi di sicurezza, verranno cifrate. L'idea iniziale è l'utilizzo di una `masterPassword` che viene richiesta all'utente in fase di registrazione. Quest'ultima sarà la chiave che, tramite un sistema crittografico simmetrico, verrà utilizzata per cifrare le password prima di salvare sul database e decifrarle quando verrà richiesto dall'utente. L'algoritmo crittografico utilizzato è: AES. In questo modo soltato chi è a conoscenza della chiave (ovvero la `masterPassword`) può riuscire a vedere i dati sensibili salvati. Abbiamo quindi ottenuto un secondo grado di sicurezza; nel caso in cui un potenziale attaccante riuscisse a trovare una collisione sull'hash della password, non sarebbe comunque in grado di visualizzare le password in chiaro.

3.2.1 Da `masterPassword` a chiave AES

Una chiave AES è una serie di byte di lunghezza fissa; dal punto di vista dell'usabilità questa soluzione non è pratica. Nasce quindi la necessità di rendere la `masterPassword` un qualcosa facile da ricordare per l'utente ma allo stesso tempo adatto per il nostro metodo di cifratura. Si è scelto di utilizzare l'algoritmo PBKDF2 che è una funzione di derivazione delle chiavi. L'utente quindi può inserire una stringa a piacere (la nostra `masterPassword`) ed essa diventerà una chiave adatta ai nostri scopi. Un prototipo di utilizzo è:

```
derivedKey = PBKDF2(masterPassword, salt, ITERATION_NUM,  KEY_LEN, prf)
```

dove:

- `ITERATION_NUM`: numero di iterazioni
- `KEY_LEN`: sarà della lunghezza richiesta dalla chiave AES
- `prf`: sta per pseudo-random function

3.2.2 Key Wrapping e soluzione al requisito cambio della master password

La soluzione discussa prima non è quella finale utilizzata nel programma. Per soddisfare il requisito della possibilità del cambio master password, l'utilizzo singolare di quest'ultima è altamente inefficiente in caso di un elevato numero di dati e utenti. L'implementazione del cambio password consisterebbe nel decriptare tutte le password salvate attraverso `oldMasterPassword` e criptarle tutte con `newMasterPassword`. La soluzione a questo problema è l'uso di una chiave intermedia (nel database è situata nella tabella *security_value* nel campo *user_encrypted_key* vedi (4.3)) e **non** è salvata in chiaro). Questa chiave intermedia serve per crifrare tutte le password prima che esse vengano salvate sul database.
La logica, in caso di cambio password, è la seguente:

1. l'utente fornisce `oldMasterPassword` e `newMasterPassword`
2. la chiave intermedia, cifrata, attualmente salvata nel database, viene decriptata utilizzando la `oldMasterPassword`
3. ottenuta la chiave intermedia in chiaro, andiamo a cifrarla con la `newMastePassword`
4. salviamo la chiave intermedia critata sul batabase

E' importante notare come in questo processo la chiave intermedia rimanga sempre inalterata, si va solo a cambiare la sua chiave di cifratura (ovvero la `masterPassword`).
Infine la password in chiaro si ottiene nel seguente modo:

1. decritto la chiave intermedia con la `masterPassword`
2. con la chiave intermedia in chiaro vado a decriptare le password e ad inviarle al client

Nel caso in cui la `masterPassword` inserita dall'utente non sia corretta anche la decrittazione della chiave intermedia sarà errata e, di conseguenza, non sarà più possibile risalire alle password in chiaro.

3.3 Gestione della sessione

La gestione della sessione avviene mediante l'utilizzo di cookie di sessione e solo in run-time; ovvero, in caso di crash e conseguente riavvio del server, tutte le sessioni andranno perse e gli utenti devono effettuare nuovamente il login.

3.4 Query parametrizzarte e protezione contro SQL Injection

Una query parametrizzata è una tecnica utilizzata per prevenire gli attacchi di tipo SQL Injection. La query, per essere eseguita, richiede l'inserzione di parametri in una posizione specifica, anzichè essere concatenati con la stringa SQL. La sintassi prevede l'uso di segnaposto o placeholder '`?`' (esempio: `query = 'SELECT * FROM ... WHERE 'nome' = ?)`.
E' importante notare come la sanificazione dell'input non è in sè un metodo di protezione contro SQL Injection in quanto nulla vieta all'utente di registrarsi con parole chiave del linguaggio SQL, questo è il motivo per la quale **non** ci si deve limitare al semplice controllo dell'input.

3.4.1 Sanificazione, controllo sull'input e admin profile

Nel nostro caso, parte del processo di sanificazione, consiste nel controllo del rispetto di determiante caratteristiche dell'input come la lunghezza e caratteri ammessi. La sanificazione delle SQL Injection è quindi inattuabile ma, in caso di possibili minacce, tutti i dati riguardanti la richiesta HTTP verranno salvati e saranno visualizzabili direttamente dall'admin (o operatore tecnico);

3.5 Progettazione del database

Riconducendoci a quanto detto precedentemente il database viene progettato e realizzato in funzione delle varie esigenze.

4 Realizzazione

Prima di cominciare è doveroso aprire una piccola parentesi per quanto riguarda lo stile utilizzato, essendo un linguaggio non tipato abbiamo utilizzato diversi stili a seconda del tipo di dato trattato:

- file: `tipofile_nomefile`
- funzioni: `nomefile_nomefunzione`
- classi: `MiaClasse`
- variabili: `miaVariabile`

4.1 Tecnologie utilizzate

La gestione delle varie richieste HTTP lato server è stata gestita utilizzando nodejs.
Per il databse abbiamo utilizzato MySQL con relativa estensione di nodejs.
Data la complessità del progetto è stata posta meno attenzione per quanto rigurda il front-end, realizzato tramite l'uso di pagine ejs e relativo codice CSS per la grafica.

4.2 Database

La struttura del database è la seguente:

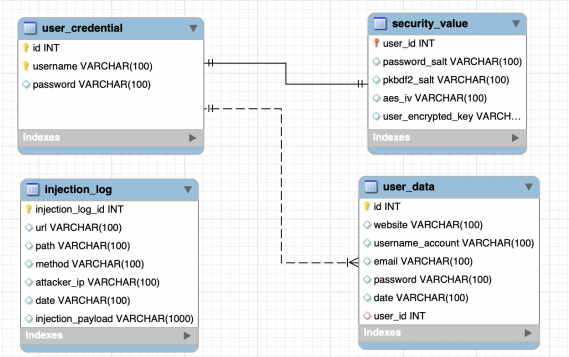

Figura 1: Database structure

Per semplicità l'admin si dovrà loggare con la tupla "admin" - "admin" e il contenuto stampato sarà direttamente quello presente nella tabella `injection_log`. Questo è il motivo per la quale quest'ultima non ha relazioni con la tabella `user_credential`.
Un aspetto importante da notare per quanto riguarda la progettazione del database è la separazione di tabelle come `user_credential` e `security_value` (quindi la separazione tra dati come hashPassword e il relativa salt). Questa pratica rende più difficile l'ottenimento di dati sensibili in caso di attacco.

4.3 Sessione

Per la gestione della sessione è stata utilizzata la libreria express-section.

4.4 Crittografia

Come detto precedentemente le tecnologie utilizzate sono: AES, BCRYPT, PBKDF2. Inoltre sono state creati due oggetti per la gestione della creazione delle chiavi e per AES. Le classi sono presenti in `/object/object_cipher` sotto il nome di `AESCipher` e `PBKDF2KeyGenerator`.

5 Conclusioni

Questo progetto è stata un'opportunità per approfondire tematiche come i cookie di sessione, le query, e la crittografia a chiave simmetrica e tutti gli aspetti relativi alla sicurezza.

5.1 Possibili innovazioni future

Le possibili innovazioni che si possono apportare al progetto sarebbero:

- Approfondita cookie e loro utilizzo. Implementazione di protezione contro attacchi come cookie-forgery.
- Protezione contro attacchi di tipo XSS (cross-site scripting).
- Aprrofondimento e implementazione delle connessioni sicure (HTTPS, TLS, etc.).
- Certificati per l'autenticazione.