Ochrona Danych

Projekt

Menadżer haseł *Sesame*

Patryk Mroczyński 126810
patryk.mroczynski@student.put.poznan.pl
Daniel Staśczak 126816
daniel.stasczak@student.put.poznan.pl
Jakub Wiśniewski 126824
jakub.t.wisniewski@student.put.poznan.pl

24 stycznia 2019

Spis treści

Spis rysunków

1 Charakterystyka ogólna projektu

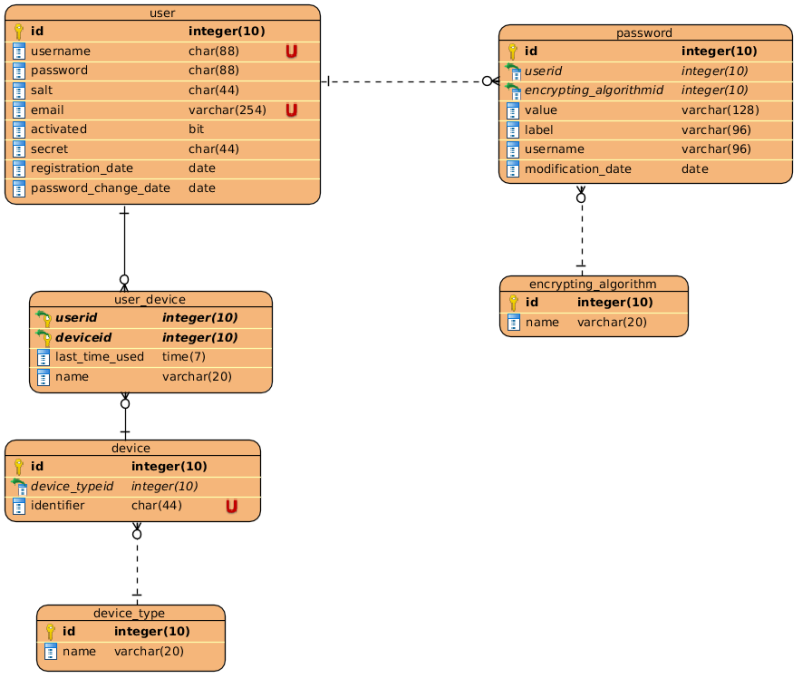
Aplikacja *Sesame* jest menadżerem haseł, ułatwiającym zarządzanie unikalnymi i bezpiecznymi hasłami w różnych serwisach. Umożliwia ona bezpieczne przechowywanie haseł na serwerze oraz zezwala na dostęp do nich tylko dla użytkownika, który zna hasło do serwisu. Użytkownik ma możliwość wprowadzenia wpisu do bazy danych, który zawiera zaszyfrowane dane używane przy logowaniu do danego serwisu oraz etykietę, która ułatwia użytkownikowi zidentyfikowanie do jakiego celu przechowywane są dane we wpisie. Szyfrowanie oraz odszyfrowanie dzieje się po stronie klienta, dzięki czemu niemożliwa jest ingerencja osób mających dostęp do serwera. Operacje te są zautomatyzowane, dzięki czemu użytkownik nie musi posiadać wiedzy dotyczącej kryptografii.

2 Architektura systemu

System oparty jest na modelu klient-serwer. Aplikacje klienckie udostępniają interfejs graficzny, umożliwiający korzystanie z serwisu osobom nietechnicznym. Na maszynie serwerowej uruchomiona jest relacyjna baza danych oraz program udostępniający API aplikacjom klienckim. API serwerowe pozwala wykonywać użytkownikom udostępnione operacje na bazie danych. W oparciu o API serwerowe, stworzone zostały API klienckie w poszczególnych językach programowaniach, które są dedykowane konkretnym platformom i które udostępniają funkcjonalności ułatwiające implementacje aplikacji klienckich.

2.1 Baza danych

Baza danych, przedstawiona na rysunku 2.1 zawiera informacje o tabelach wykorzystywanych w serwisie. Tabele user, password oraz device stanowią bazę dla głównej funkcjonalności w serwisie, a wszystkie pozostałe tabele są tabelami pomocniczymi — udostępniają dodatkowe funkcjonalności, zwiększające atrakcyjność serwisu i bezpieczeństwo przetrzymywanych informacji wrażliwych. Tabela user zawiera informacje o użytkowniku, niezbędne do prawidłowego i bezpiecznego funkcjonowania aplikacji oraz umożliwiające użytkownikom korzystanie z funkcjonalności systemu. Tabela password zawiera wpisy zawierające zaszyfrowane hasła dla konkretnych użytkowników, a tabela device umożliwia korzystanie z aplikacji tylko zweryfikowanym przez użytkownika urządzeniom. Tabela encrypting_algorithm umożliwia zdefiniowanie nowych metod szyfrowania haseł, nie tracąc jednocześnie informacji o metodach stosowanych przy szyfrowaniu poprzednich haseł. Informacja o wykorzystanym algorytmie znajduje się w tabeli password jako klucz obcy. Tabela device_type umożliwia przechowywanie informacji o typie urządzenia. Nie jest ona obecnie wykorzystywana, ale zbierane przez nią informacje mogą posłużyć do rozbudowania serwisu o dodatkowe funkcjonalności takie jak autoryzacja dwuetapowa.



Rysunek 2.1: Model bazy danych

2.2 Serwisy

Aplikacja serwerowa udostępnia szereg serwisów, które są wykorzystywane do komunikacji pomiędzy klientem i serwerem. Serwisy stanowiące trzon funkcjonalności systemu to:

- AuthService — System umożliwiający użytkownikom utworzenia konta w serwisie, służący do aktywacji konta oraz udostępniający możliwość wygenerowania tokenów JWT, używanych do dalszej autoryzacji w serwisie.
- UserService — Pozwala na pobranie informacji o użytkowniku takich jak adres email, data rejestracji itp. Umożliwia zmianę poszczególnych danych użytkownika (np. głównego hasła). Do wykonywania operacji na tym serwisie wymagany jest token uwierzytelniający JWT.
- PasswordService — Udostępnia metody pozwalające na zarządzanie przetrzymywanymi hasłami w serwisie. Pozwala pobrać listę etykiet haseł, dodać nowe hasło, usunąć hasło oraz pobrać hasło na podstawie etykiety. Serwis ten wymaga użycia tokenu JWT w celach autoryzacyjnych.

3 Bezpieczeństwo

W rozdziale opisane zostały wszystkie zastosowane w aplikacji aspekty bezpieczeństwa. Wymienione zostały wszystkie zastosowane mechanizmy na każdym etapie działania systemu.

Mechanizmem zabezpieczającym ujawnienie hasła użytkownika poprzez wyciek informacji z bazy danych jest wykorzystanie funkcji PBKDF2, która zapewnia większe bezpieczeństwo od używania funkcji skrótu. Została ona zaprojektowana do utrudnienia prób ataków brute-force oraz słownikowych. Wydłuża ona czas potrzebny na obliczenie pojedynczego skrótu w stopniu zależnym od zastosowanej liczby iteracji, a wykorzystanie soli, która jest indywidualna dla każdego użytkownika i dodawana do każdego hasła, zapewnia różne skróty dla identycznych haseł różnych użytkowników. Dzięki temu zmniejszona jest skuteczność tablic tęczowych — atakujący musiałby tworzyć osobne tablice dla każdej soli.

Dzięki użyciu protokołu HTTPS, przechwycenie przesyłanych danych jest bardzo trudne, ale pozostaje problem widoczności nieprzetworzonego hasła po stronie serwera. Aby zabezpieczyć użytkowników przed wykorzystaniem tego faktu przez atakujących, wykorzystana zostaje ponownie funkcja PBKDF2, tym razem po stronie klienta. Jako sól użyty zostaje wynik funkcji skrótu nieprzetworzonej nazwy użytkownika w połączeniu z nieprzetworzonym hasłem, a liczba iteracji jest mniejsza niż po stronie serwerowej, aby użytkownik nie odczuł wykonywanych operacji. Sama nazwa użytkownika zostaje przesłana na serwer jako wynik funkcji skrótu.

Aby zmniejszyć prawdopodobieństwo przechwycenia pakietu zawierającego hasło użytkownika, wykorzystane zostały tokeny JWT. Dzięki temu aplikacja kliencka po poprawnym logowaniu otrzymuje token zawierający informacje niezbędne do wykonywania kolejnych zapytań (takie jak skrót nazwy użytkownika czy czas wygaśnięcia tokenu). Całość tokenu jest podpisana za pomocą algorytmu kryptograficznego HMAC, wykorzystującego sekret serwerowy, co zapewnia autentyczność i nienaruszalność tych danych. Dodatkowo dodawany jest kolejny sekret (przechowywany w bazie danych), który jest unikalny dla każdego użytkownika, dzięki czemu, po jego wykryciu sekretu serwerowego przez atakującego, niemożliwe jest fałszowanie podpisu dla każdego użytkownika. Sekret serwerowy jest zmieniany co określony odcinek czasu.

Każdorazowe logowanie na nowym (nieautoryzowanym wcześniej) urządzeniu powoduje wysłanie wiadomości e-mail do użytkownika, aby ten mógł zweryfikować czy akcja została wykonana na jego urządzeniu, czy jest to próba włamania. Dzięki zastosowaniu tego mechanizmu, dostęp do danych użytkownika otrzymują tylko urządzenia należące do niego. Aby umożliwić takie zabezpieczenie, aplikacja kliencka musi wysyłać skrót swojego identyfikatora, którym

może być np. adres MAC komputera czy fingerprint przeglądarki. Dopiero po poprawnym zweryfikowaniu hasła i identyfikatora, generowany jest token JWT, którego czas wygaśnięcia to krótki odcinek czasu (dzięki czemu nawet w przypadku przechwycenia tokenu, atakujący można korzystać z niego tylko przez chwilę).

Po każdej interakcji aplikacji klienckiej z serwerem, zostaje dodany wpis do logów, który zawiera informacje o użytkowniku, adresie IP, akcji, która została wykonana oraz o powodzeniu akcji. Dzięki temu zapewniona jest niezaprzeczalność i rozliczalność, co pozwala na śledzenie podejrzanych aktywności oraz przeciwdziałanie im.

Hasła zapisane przez użytkownika, przechowywane na serwerze są w postaci zaszyfrowanej symetrycznie. Użytkownik może wybrać czy do odszyfrowania hasła chce używać zmodyfikowanego przez aplikację kliencką hasła do serwisu czy użyć osobnego klucza, podanego przez użytkownika. Całość szyfrowania i deszyfrowania odbywa się w aplikacji klienckiej, dzięki czemu klucz nigdy nie wydostaje się poza urządzenie użytkownika.

Do generowania soli oraz sekretów po stronie serwera używane są algorytmy wykorzystujące kryptograficznie bezpieczne źródło losowości (np. `/dev/urandom` w przypadku systemów opartych o UNIX lub `CryptGenRandom` w przypadku systemów Windows).

4 Wymagania

W rozdziale zostały opisane wymagania funkcjonalne oraz niefunkcjonalne serwera, aplikacji klienckich oraz komunikacji między nimi wraz z podziałem na aktorów.

4.1 Wymagania funkcjonalne

W tej sekcji przedstawione zostały wymagania funkcjonalne z podziałem na dwóch aktorów — użytkownika niezalogowanego i użytkownika zalogowanego.

- Niezalogowany użytkownik
  - Możliwość utworzenia konta w serwisie.
  - Możliwość zalogowania do serwisu za pomocą hasła głównego i nazwy użytkownika.
- Zalogowany użytkownik
  - Możliwość wylogowania.
  - Możliwość edycji w opcjach konta adresu email oraz hasła głównego.

  - Możliwość dodania nowego przechowywanego hasła do serwisu wraz z informacjami dodatkowymi — etykietą i nazwą użytkownika.
  - Możliwość usunięcia hasła przechowywanego.
  - Możliwość edycji hasła przechowywanego oraz informacji dodatkowych.
  - Możliwość wyświetlenia etykiet przechowywanych haseł.
  - Możliwość wyboru etykiety i przekopiowania do schowka hasła lub nazwy użytkownika dla danego wpisu.

4.2 Wymagania niefunkcjonalne

W wymaganiach niefunkcjonalnych opisane są technologie jak i techniki związane z budową aplikacji klienckiej oraz serwerowej.

- Aplikacja serwerowa napisana w języku Python3.
- Aplikacja serwerowa umożliwiająca administratorowi wybór systemu relacyjnej bazy danych pomiędzy PostgreSQL, MySQL oraz SQLite3.
- Aplikacja serwerowa udostępnia publiczne API, na podstawie którego budowane mogą być aplikacje klienckie.
- Aby użytkownik mógł korzystać z aplikacji musi się poprawnie zalogować na założone w serwisie konto.
- Dane użytkownika są przechowywane w bazie danych.
- Główne hasło użytkownika przechowywane jest jako wynik funkcji rozciągania klucza PBKDF2 z wykorzystaniem algorytmu SHA512 i 100000 iteracji.
- Wszelkie wykorzystywane źródła losowe po stronie serwerowej powinny być kryptograficznie bezpieczne.
- Hasło użytkownika powinno posiadać minimum 16 znaków długości.
- System autoryzacji oparty jest o tokeny JWT.
- Aplikacje klienckie powinny udostępniać wszystkie najważniejsze funkcjonalności systemu.
- API stworzone jest zgodnie z architekturą REST, w formie serwisów.
- Hasła przechowywane w serwisie są szyfrowane za pomocą wybranej metody szyfrowania symetrycznego.
- Szyfrowanie i deszyfracja haseł odbywa się wyłącznie po stronie aplikacji klienckiej.

5 Narzędzia, środowiska, biblioteki

Serwer działa w oparciu o system operacyjny Ubuntu Server 18.04 LTS. Do zarządzania danymi wykorzystywana jest relacyjna baza danych PostgreSQL 10 Mikroserwisy działające jako aplikacja serwerowa napisane są w języku Python 3.7 wraz z frameworkiem webowym CherryPy oraz biblioteką udostępniającą funkcjonalności ORM i łączności z bazą danych Peewee.

Pierwsza aplikacja kliencka to aplikacja desktopowa napisana w języku Python 3.7, testowana na systemach operacyjnych Windows 10 oraz wybranych dystrybucjach Linuxa: Ubuntu 18.04 LTS i Antergos Linux. Biblioteką wykorzystaną w projekcie, która udostępnia funkcje ułatwiające tworzenie interfejsów użytkownika jest Tkinter.

Dodatkowe narzędzia wykorzystane przy tworzeniu aplikacji to Visual Studio Code, Vim, DBeaver Community Edition 5 oraz Postman.

Narzędzia, które zostały wykorzystane do wspierania pracy zespołowej to Discord do komunikacji głosowej podczas pracy, system kontroli wersji Git wraz z repozytorium na stronie GitHub.com, TeX Live do tworzenia dokumentacji oraz Visual Paradigm do tworzenia schematów oraz diagramów.

6 Możliwości rozwoju

W ty rozdziale omówione zostały możliwe kierunki rozwoju aplikacji.

- Wprowadzenie opcjonalnej autoryzacji dwuetapowej, umożliwiającej użytkownikom zwiększenie bezpieczeństwa przechowywanych w serwisie haseł. System ten mógłby zostać zaimplementowany w formie aplikacji mobilnej dostępnej w systemach Android i iOS, które po ówczesnym zalogowaniu i oznaczeniu urządzenia jako zaufanego mogłyby generować kilkucyfrowe kody autoryzacyjne.
- Udostępnienie większej ilości oficjalnych aplikacji klienckich, które mogłyby być dostępne jako aplikacje mobilne na systemy Android i iOS, rozszerzenie w przeglądarkach czy aplikacja przeglądarkowa.
- Ustalenie jednolitej szaty graficznej aplikacji klienckich i elementów ułatwiających rozpoznanie systemu takich jak logo czy kolorystyka.
- Wprowadzenie do Client API funkcjonalności sparametryzowanego generowania silnych haseł. Użytkownik końcowy mógłby ustalić m.in. pożądaną długość hasła i pulę znaków użytych w trakcie generowania (spośród małych i dużych liter, cyfr i znaków specjalnych).
- Wprowadzenie do Client API funkcjonalności analizy siły już zapisanych haseł.