Bezpieczeństwo medycznych systemów informacyjnych
2023L

Szablon aplikacji WEB wirtualnej przychodni- uwierzytelnienie, autoryzacja i przechowywanie sesji z minimalnym wykorzystaniem cookies

Autorzy:
Zuzanna Popławska
Dominika Powałka
Grzegorz Molak

Spis treści

Cel projektu

Aplikacje internetowe są to swego rodzaju programy komputerowe umieszczone na serwerze, które za pośrednictwem połączenia internetowego wymieniają dane pomiędzy siecią komputerową a hostem użytkownika komputera poprzez przeglądarkę internetową. Zdalny dostęp do zasobów pozwala na coraz bardziej kreatywne formy ataków wymusza na programistach ciągłe dbanie o bezpieczeństwo aplikacji webowych. Celem takich ataków jest zazwyczaj uzyskanie nieautoryzowanego dostępu do danych lub sparaliżowanie działania usługi. Analiza ryzyka teleinformatycznego jest istotnym krokiem w projektowaniu i wdrażaniu aplikacji, aby zapewnić odpowiednie zabezpieczenia i minimalizować potencjalne zagrożenia.

Współczesne przeglądarki internetowe

Współczesne przeglądarki oferują zaawansowane mechanizmy przechowywania sesji, takie jak pliki cookie, magazyny lokalne (local storage) i sesje HTTP. Pliki cookie są niewielkimi plikami tekstowymi przechowywanymi na urządzeniu użytkownika, które zawierają informacje o sesji i preferencjach użytkownika. Magazyny lokalne umożliwiają przechowywanie większych ilości danych, takich jak ustawienia aplikacji czy dane użytkownika. Sesje HTTP są mechanizmem przechowywania informacji o sesji po stronie serwera.

Ważne jest, aby przeglądarki implementowały odpowiednie zabezpieczenia, aby zapewnić bezpieczeństwo tych mechanizmów. Na przykład, pliki cookie powinny być odpowiednio oznaczane i zarządzane, aby uniemożliwić ich nieuprawnione wykorzystanie przez strony trzecie. Mechanizmy przechowywania danych powinny być chronione poprzez stosowanie odpowiednich protokołów szyfrowania, takich jak HTTPS, aby zapobiec przechwyceniu danych przez nieautoryzowane osoby trzecie. W celu nadawanie i weryfikacja uprawnień do danych przeglądarki wykorzystują mechanizmy autoryzacji i uwierzytelniania, takie jak certyfikaty SSL/TLS i protokoły uwierzytelniania, aby zapewnić bezpieczeństwo.

Realizacja systemu webowego, implementacja zabezpieczeń

Backend – https://gitlab-stud.elka.pw.edu.pl/gmolak/bemsi.git
Frontend – https://gitlab-stud.elka.pw.edu.pl/zpoplaws/bemsi-react.git
Baza danych – MySQL

Instrukcja jak kompilować i uruchamiać znajduje się w plikach README.md

Analiza ryzyka i przyjęte modele zabezpieczeń

Podczas realizacji projektu aplikacji webowej Wirtualna Przychodnia konieczne było przeprowadzenie analizy w celu stworzenia i zaimplementowania najbardziej optymalnych mechanizmów zapewniających bezpieczeństwo danych użytkowników (często wrażliwych, takich jak dane osobowe lub historia choroby). Główne i najpoważniejsze zagrożenia dotyczące aplikacji Wirtualna Przychodnia to ataki związane z uwierzytelnianiem i autoryzacją skutkujące ujawnienie poufnych danych medycznych osobom do tego nieuprawnionym. W celu ochrony przed nimi zaimplementowaliśmy cztery niezbędne zabezpieczenia opisane poniżej.

1. Protokół HTTPS

Protokół HTTPS (Hypertext Transfer Protocol Secure) jest wykorzystywany do bezpiecznej komunikacji między klientem a serwerem, zapewnia poufność, integralność i uwierzytelnianie danych. Komunikacja rozpoczyna się wysłaniem zaszyfrowanego żądania HTTPS przez przeglądarkę. Następnie serwer prezentuje certyfikat SSL/TLS , który jest weryfikowany po stronie klienta. Po udanej weryfikacji negocjowany jest klucz szyfrowania.

Podczas pisania aplikacji w celu zastosowania komunikacji po HTTPS wygenerowaliśmy samopodpisany certyfikat. Przeglądarki nie uznają takich certyfikatów za wiarygodne, co może prowadzić do ostrzeżeń bezpieczeństwa dla użytkowników. W produkcyjnym środowisku należałoby uzyskać certyfikat podpisany przez zaufaną instytucję.

2. Bezpieczeństwo haseł

Hasła przechowywane są w bazie danych w postaci zaszyfrowanej. W tym celu wykorzystujemy funkcję jednokierunkową **SHA256**, która pozwala na obliczenie skrótu. Takie proste rozwiązanie jest podatne na różnego rodzaju ataki, aby lepiej się przed nimi chronić wykorzystaliśmy algorytm **PBKDF2**, którego ważnymi parametrami są liczba iteracji, unikalna sól oraz długość wyniku. Sól generowana w naszym kodzie jest 16-bajtowym ciągiem losowych znaków, który jest dodawany do hasła w takcie obliczeń. W każdej iteracji hasło jest poddawane funkcji haszującej z wynikiem poprzedniej iteracji jako dodatkowymi danymi. Po zakończeniu kodowania do bazy danych zapisujemy uzyskany wynik wraz z parametrami potrzebnymi do weryfikacji w postaci: 'liczba_iteracji':'sól':'zaszyfrowane_hasło'. Podczas weryfikacji hasła szyfrujemy wprowadzoną przez użytkownika wartość, przy użyciu tych samych parametrów co przy rejestracji i wynik porównujemy z zapisanym w bazie danych.

Większa liczba iteracji powoduje większe opóźnienie przy generowaniu klucza, co utrudnia ataki **brute force**. Ciężki algorytm haszujący może jednak spowodować Denial of Sevice. Unikalna sól dla każdego hasła zapobiega atakom z wykorzystaniem **tęczowych tablic**, które są prekalkulowanymi zestawami haszów dla popularnych haseł. W zastosowanym rozwiązaniu nawet jeśli dwaj użytkownicy mają takie same hasła, to przechowywane w bazie danych skróty będą różniły się.

Generowane bazowe hasła mają 12 znaków i są ciągiem losowo wygenerowanych liter i cyfr. Dzięki temu spełniają wymogi bezpieczeństwa. Są one jednak ciężki do zapamiętania i wprowadzania, dlatego też każdy użytkownik ma opcję jego zmiany. Nowe hasło musi mieć również minimum 12 znaków.

3. Autoryzacja przy pomocy tokenów JWT

Autoryzacja za pomocą **tokenu JWT** została zrealizowana przy pomocy biblioteki JJWT dostępnej w repozytorium: https://github.com/jwtk/jjwt .

Osobno wygenerowany 'sekret' zostaje zakodowany w formacie Base64 i przechowywany w pliku konfiguracyjnym aplikacji. Nie opuszcza on nigdy serwera. Token przechowuje w nagłówku informację o typie tokenu(JWT) oraz o rodzaju algorytmu służącego do stworzenia sygnatury(**HS256** czyli HMAC using SHA256). W ciele(payload) tokenu znajduje się **login użytkownika**, do którego przypisany jest token oraz jego data ważności. Token jest ważny 5 minut od jego utworzenia. Ostatni człon tokenu stanowi jego podpis(signature), utworzony na podstawie danych zawartych w tokenie oraz przechowywanego sekretu. Rolą podpisu jest zapewnienie autentyczności danych, uniemożliwia podmianę danych(np. loginu), ponieważ w takim wypadku serwer wykryje, że treść tokenu jest niezgodna z jego podpisem. Niemożliwa jest również autoryzacja przy użyciu nagłówka ze sprecyzowanym algorytmem "None", która miałaby omijać sprawdzanie podpisu.

Aby zachować token podczas korzystania z aplikacji, skorzystaliśmy z **ciasteczka** do przechowywania tokenu. Aby uniemożliwić ataki polegające na jego wykradnięciu(np. **ataki XSS**) ciasteczko takie musi być **HttpOnly**, co oznacza, że niemożliwy jest dostęp do niego z poziomu JavaScriptu. Aby zabezpieczyć jego transmisję między klientem a serwerem ustawiona zostaje również flaga **Secure**, umożliwiająca przesyłanie ciasteczka tylko przez SSL/HTTPS.

Ważną kwestią jest też zapewnienie, aby w niektórych przypadkach token, pomimo poprawnej zawartości oraz podpisu, który nie jest przedawniony nie był autoryzowany przez serwer. Przykładowymi sytuacjami, w których tak się dzieje jest np. wylogowanie użytkownika bądź odświeżenie tokenu, to jest przydzielenie użytkownikowi nowego tokenu przez utratą ważności poprzedniego. W tym celu, przy akcji wylogowania bądź odświeżenia tokenu, dotychczas używany token trafia do tabeli **tokenów unieważnionych** wraz z datą jego ważności. W tym momencie zaprezentowanie serwerowi poprawnego tokenu, który jednak znajduje się na liście tokenów unieważnionych, nie umożliwi autoryzacji. Aby nie przeciążać bazy danych, unieważnione tokeny mogą być z niej usunięte po upłynięciu ich daty ważności, kiedy i tak nie umożliwiają poprawnej autoryzacji.

4. Kontrola dostępu do zasobów

Kontrola dostępu do zasobów zrealizowana została przy użyciu przypisanych do użytkowników ról oraz sprawdzaniu „posiadania” zasobu.

Kontrola oparta na rolach

Każdy użytkownik posiada przydzieloną do siebie rolę, odpowiadającą jednej lub wielu podstawowym rolom, a zapisaną w postaci liczby całkowitej. Jest ona sumą liczb odpowiadającym rolom: pacjenta, lekarza, personelu pomocniczego i administratora. Każda z tych podstawowych liczb jest kolejną potęgą liczby dwa, co umożliwia proste sprawdzenie roli przy pomocy iloczynu logicznego. Przykładowo, jeżeli użytkownik ma być jednocześnie lekarzem, przypisana mu zostanie rola 1 + 2 = 3, a przynależność do każdej z tych ról można sprawdzić przy użyciu rola & 1 == 1 i rola & 2 == 2. W przypadku, gdy użytkownik nie posiada danej roli, wynik iloczynu logicznego wyniesie 0. W zależności od roli użytkownicy mają różny dostęp do zasobów(uprawnienia). Metoda ta umożliwia w prosty sposób kontrolę możliwości rejestracji nowych użytkowników, czy umożliwienie zapisy na wizyty personelowi pomocniczemu, a zablokowanie tej możliwości lekarzowi. Sytuacja jednak znacznie utrudnia się w przypadku, gdy użytkownik posiada więcej niż jedną rolę.

Kontrola oparta na posiadaniu zasobu

Pojęcie posiadania jest zależne od rodzaju tego zasobu. Z tego powodu bardzo trudne jest wymyślenie zasad, które w prosty sposób umożliwiły obsłużenie wszystkich możliwych przypadków. Z tego powodu zasady dla każdego zasobu muszą zostać wymyślone osobno. Bycie właścicielem zasobu profilu użytkownika wiąże się z bycia dokładnie tym użytkownikiem. Dostęp do wizyty jest zależny od tego, czy jest na nią ktoś zapisany czy nie. Jeżeli tak, dostęp do niej mają biorący w niej udział lekarz i pacjent oraz, pod warunkiem że wizyta jeszcze się nie wydarzyła – personel pomocniczy(np. w celu jej odwołania w imieniu pacjenta). Jeżeli wizyta jest wolna, dostęp do niej ma dowolny pacjent, personel pomocniczy oraz przeprowadzający ją lekarz. Zapis na wizytę możliwy jest przez dowolnego pacjenta i personel pomocniczy. Odwołanie wizyty jest możliwe przez pacjenta zapisanego na tą wizytę bądź personel pomocniczy. Historia wizyt danego użytkownika jest zawsze dostępna dla tego samego użytkownika. W przypadku próby przejrzenia historii wizyt innego użytkownika jest to możliwe tylko i wyłącznie w przypadku lekarza i ujawnią się jedynie wizyty, w których on sam brał udział. Kalendarz wizyt jest dostępny również dla personelu pomocniczego,

ponieważ nie posiada on jeszcze notatki z wizyty, w związku z czym nie posiada żadnych wrażliwych danych medycznych – chociaż możliwe jest odgadnięcie przypadłości pacjenta ze względu na tożsamość lekarza. Większą prywatność dałoby się uzyskać, gdyby informacja o wizycie zawierała jedynie np. identyfikator lekarza, a nie jego imię, nazwisko oraz specjalizację.

Testy

Poprawność działania funkcjonalności API była weryfikowana przy pomocy Postmana poprzez wysłanie żądania i weryfikację rezultatów czy pokrywają się z oczekiwanymi. Część z tych testów zostanie zaprezentowana podczas oddania projektu.

Proponowane rozszerzenia modelu zabezpieczeń

- Skonfigurowanie firewall w celu blokowania ruchu nadmiernego lub niepożądanego. Można skorzystać listy białej lub listy czarnej adresów IP, aby kontrolować, który ruch jest akceptowany lub blokowany. W następnym kroku należałby monitorować ruch sieciowy i wykrywać nieprawidłowości.

- Ograniczenie ilości nieudanych prób zalogowania na jednego klienta. Wymuszanie nie tylko długich, ale też silnych (unikatowe/niepopularne) haseł. Pozwoliło by to jeszcze skuteczniej chronić się przed atakami związanymi z uwierzytelnianiem.

- Co można było zrobić lepiej – w trakcie logowania, osoba posiadająca wiele ról musi wybrać jako kto się loguje, uprościłoby to znacznie logikę biznesową aplikacji oraz uniemożliwiłoby przydzielenie pojedynczemu zalogowanemu użytkownikowi zbyt wiele uprawnień naraz. Wymagałoby to jednak przechowywanie roli w tokenie użytkownika.