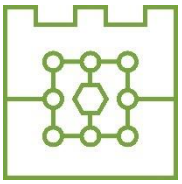
Politechnika Krakowska
im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji



Michał Florczak

Numer albumu:119692

Analiza bezpieczeństwa oprogramowania Keycloak

Keycloak security analysis

Praca magisterska
na kierunku Informatyka
specjalność Cyberbezpieczeństwo

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Kraków, 2023

Spis treści

Wstęp 4

1.1 Motywacja pracy 4

1.2 Cel pracy 4

1.3 Zakres pracy 4

1.4 Metodyka pracy 5

I Część teoretyczna 6

2. Podstawy Teoretyczne 6

2.1 Architektura mikroserwisów 6

2.2 Projektowanie współczesnych systemów informatycznych 7

2.3 Architektura rozwiązania 11

2.4 Rodzaje podatności 12

2.5 Metody autoryzacji 17

2.6 Metody tworzenia bezpiecznego frontendu 18

2.7 Metody tworzenia bezpiecznego backendu 18

2.8 Metody autoryzacji JWT i Session 19

2.9 Przegląd narzędzi do testowania 22

II Część praktyczna 25

3. Implementacja 25

3.1 Zbieranie wymagań 25

3.2 Przygotowanie infrastruktury 25

3.3 Analiza aplikacji Keycloak z domyślną konfiguracją 34

3.4 Polityki Haseł w Keycloak 43

3.5 Brute force 53

3.6 SQL Injection 55

3.7 XSS 59

Wnioski 61

Spis rysunków 62

Spis listingów 64

Bibliografia 65

Wstęp

Streszczenie

Część teoretyczna pracy prezentuje sposób zabezpieczania aplikacji, rodzaje podatności oraz narzędzia jakich użyto do przeprowadzenia testów. Natomiast część praktyczna przedstawia sposoby możliwych konfiguracji Keycloaka, które zwiększają bezpieczeństwo i ataki na tą aplikacje w celu wykrycia potencjalnych luk w zabezpieczeniach. Aplikacja Keycloak jest odporna na ataki i może być z powodzeniem używana do zabezpieczania oprogramowania.

Abstract

The theoretical part of the paper presents the method of securing applications, types of vulnerabilities, and the tools used for preparing tests. The practical part, on the other hand, showcases various configurations of Keycloak that enhance security and show attacks on the application to identify potential lack of security. Keycloak is a safe application that can be successfully used for software security.

1.1 Motywacja pracy

Olbrzymi postęp technologiczny znacząca przyczynił się do podniesienia standardu życia ludzkości. Niestety wraz z rozwojem pojawiły się nowe zagrożenia cybernetyczne. Problem bezpieczeństwa jest na tyle istotne, że powstały specjalne organizacje rządowe, które dbają o infrastrukturę krytyczną. Przykładem organizacji jest Wojsko Obrony Cyberprzestrzeni [1].

Chcąc zabezpieczyć aplikacje przed nieuprawnionym lub nieautoryzowanym dostępem wiele firm szuka rozwiązania gotowego do użycia. Jednak ciężko jest sprawdzić czy wybrany mechanizm jest rzeczywiście bezpieczny, ponieważ w sieci nie istnieją powszechnie dostępne zasoby opisujące poziom bezpieczeństwa dostępnych rozwiązań. Najbardziej popularnym rozwiązaniem open source [2] jest Keycloak [3]. Autor pracy chce użyć tego oprogramowania w własnym startupie, jeżeli rozwiązanie to jest bezpieczne i zabezpiecza system przed wyciekiem prywatnych danych.

1.2 Cel pracy

Celem sprawdzenia czy oprogramowanie Keycloak jest bezpieczna stworzono infrastrukturę, która składa się z dwóch prostych aplikacji frontend i backend. Cel pracy obejmuje sprawdzenie, czy oprogramowanie Keycloak jest bezpieczne i nadaje się do użycia w startupie.

1.3 Zakres pracy

Zaimplementowano mechanizmy bezpieczeństwa dostarczone przez Keycloak w tych aplikacjach i kolejno wykonano sprawdzenie bezpieczeństwa na znane podatności.

Przetestowano, czy aplikacja jest podatna na ataki typu injection i brute force [4]. Wykonano również wiele testów manualnych szukających luk w systemie.

1.4 Metodyka pracy

Pierwsze analizy przeprowadzono przy pomocy narzędzi jak i również manualnych testów. Znalezione podatności, jeżeli miały znaczący wpływ na bezpieczeństwo aplikacji próbowano wyeliminować, poprzez dostarczenie odpowiedniej konfiguracji dla Keycloak.

Zanim przystąpiono do pierwszy analiz, konieczne było utworzenie infrastruktury. W tym celu stworzono dwie aplikacje frontend i backend. Backend został napisany w języku Java i frameworku Spring Boot [5]. Natomiast w frontend użyto VueJs [6]. Stworzono obrazy docker'owe [7] aplikacji i tak przygotowaną infrastrukturę uruchomiono w celu przeprowadzenia testów bezpieczeństwa. Jednym z narzędzi analizujących był OWASP ZAP [8], które zostało stworzone do skanowania aplikacji web. Testy wykonywano na lokalnym komputerze.

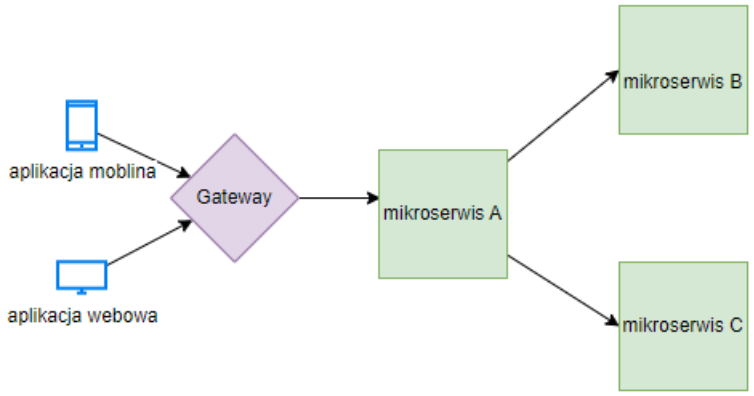
I Część teoretyczna

2. Podstawy Teoretyczne

Część ta wyjaśnia aspekty teoretyczne, które będą wykorzystane w części praktycznej. Przedstawia sposoby rozwoju współczesnego oprogramowania. Prezentuje różnicę pomiędzy rożnymi metodami uwierzytelnienia. Koniec części teoretycznej zawiera informacje o podatnościach i bibliotekach jakie zostały użyte podczas testów.

2.1 Architektura mikroserwisów

Architektura ta polega na podzieleniu, dziedziny problemu na domeny i poddomeny [9]. Kolejno w biorąc pod uwagę zależność między domenami próbuje się wydzielić mikroserwisy, które będą implementować logikę biznesową. Założenie jest, że mikroserwisy powinny być małe, przez co mogą być w łatwy sposób rozwijane. Podejście to pozwala wytwarzać oprogramowanie przez różne zespoły programistów, które są w stanie programować, testować i instalować oprogramowanie jednocześnie nie będąc zależnym od innych zespołów. Mikroserwisy przez to, że są małe szybciej się instalują na serwerach w porównaniu do wielkich aplikacji monolitycznych. Dlatego poprawnie wykorzystanie mikroserwisów skraca czas dostarczania nowych funkcjonalności do klienta. Testowanie również jest łatwiejsze, ponieważ mikroserwisy z natury są niezależne, przez co można testować poszczególne części systemu, bez konieczności uruchamiania wszystkich jego części składowych. *Rys. 1* prezentuje architekturę mikroserwisów, na której widać, że z punktu widzenia mikroserwisu A, nie ma znaczenia, czy komunikuje się z nim aplikacja mobilna czy też webowa.



Rys. 1 Architektura mikroserwisów

2.2 Projektowanie współczesnych systemów informatycznych

Projektując współczesne systemy informatyczne klasy enterprise bardzo często określa się granicę kontekstów (ang. Bounded Context). System zaprojektowany w ten sposób może być rozwijany niezależnie przez wiele zespołów programistycznych jednocześnie. Jedyne konieczne wymaganie jest zdefiniowanie miejsc w których mikroserwisy pomiędzy różnymi kontekstami będą się komunikować. Zidentyfikowanie tych miejsc powoli ustalić kontrakty w jaki sposób serwisy będą wymieniać informacje pomiędzy sobą. *Listing 1* prezentuje przykładowy kontrakt, który obrazuje w jaki sposób mikroserwis będzie się komunikował ze światem zewnętrznym.

Listing 1 Kontrakt pomiędzy dwoma interfejsami

```
openapi: 3.0.0
info:
  title: Mikroserwis zarządzający użytkownikami
  version: 1.0.0
servers:
  - url: https://api.example.com/v1
paths:
  /users:
    get:
      summary: Pobierz wszystkich użytkowników
      operationId: getUsers
      responses:
        '200':
          description: Operacja zakończona sukcesem zwróci listę użytkowników
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User'
    post:
      summary: Stwórz nowego użytkownika
      operationId: createUser
      requestBody:
        required: true
        content:
          application/json:
```

```
          schema:
            $ref: '#/components/schemas/User'
    responses:
      '201':
        description: Operacja zakończona sukcesem zwróci stworzonego użytkownika
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
/users/{userId}:
  get:
    summary: Pobierz użytkownika na podstawie jego numeru Id
    operationId: getUserById
    parameters:
      - name: userId
        in: path
        description: Id użytkownika
        required: true
        schema:
          type: string
    responses:
      '200':
        description: Operacja zakończona sukcesem zwróci jednego użytkownika
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
  put:
    summary: Zaktualizuj informacje o użytkowniku na podstawie jego Id
    operationId: updateUserById
    parameters:
      - name: userId
        in: path
        description: Id użytkownika
        required: true
        schema:
          type: string
    requestBody:
      required: true
```

```
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      responses:
        '200':
          description: Operacja aktualizacji danych zakończona sukcesem zwróci
zaktualizowane dane użytkownika
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'
    delete:
      summary: Usuń użytkownika na podstawie jego Id
      operationId: deleteUserById
      parameters:
        - name: userId
          in: path
          description: Id użytkownika
          required: true
          schema:
            type: string
      responses:
        '204':
          description: Użytkownik został usunięty poprawnie z bazy danych
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: string
          format: uuid
          description: Id użytkownika
        name:
          type: string
          description: Imię użytkownika
        email:
          type: string
```

```
        format: email

        description: Email użytkownika
```

Programista przy pomocy odpowiedniego narzędzia, może wygenerować kod kontrolerów. Znacząco przyśpiesza to pracę, ponieważ z tego kontraktu można wygenerować kod dla aplikacji serwerowej, mobilnej oraz webowej. Określenie kontraktów da możliwość pracy równoległej programistów, co z kolei skróci czas wytwarzania oprogramowania. Stosując metodę iteracyjną najpierw powstałby jeden mikroserwis, który by udostępniał swoje zasoby, a następni programiści zaczęliby tworzyć drugie rozwiązanie korzystające z API pierwszego serwisu. Poniższy przykład pokaże czas wytwarzania oprogramowania przy użyciu tych dwóch rożnych sposobów, przy następujących założeniach.

Biorąć pod uwagę poziom skomplikowania system, zespół określił, że jest w stanie stworzyć kompletne rozwiązanie w ciągu 2 miesięcy.

Metoda bez zdefiniowanych kontekstów:

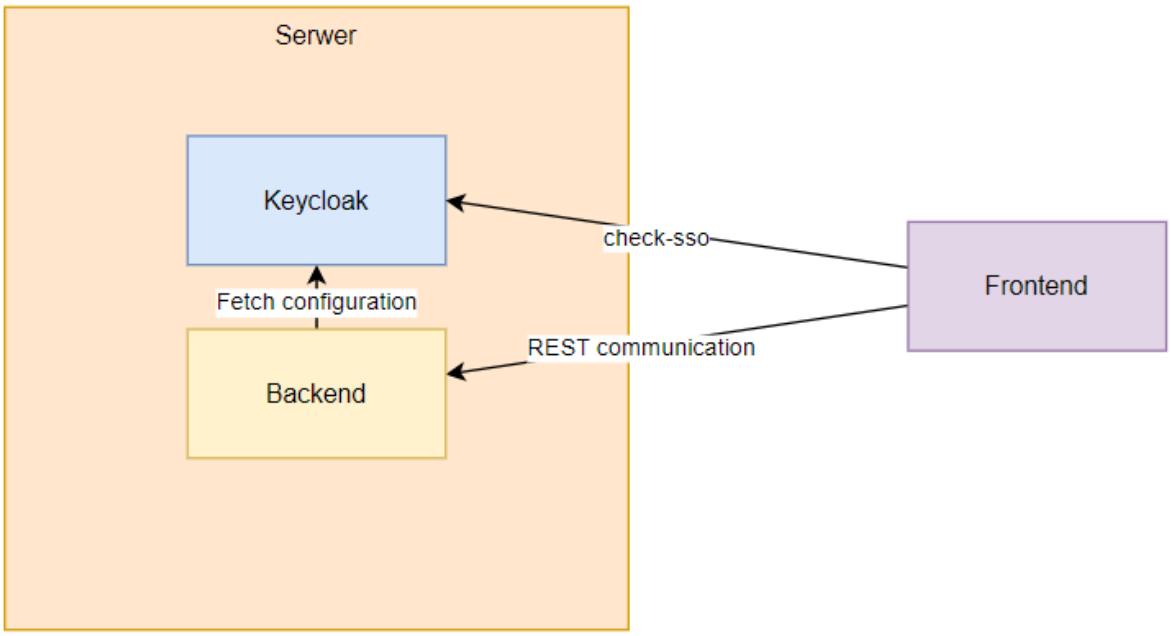
Zespół nie zdefiniował kontekstów, przez co nie było możliwości podzielenia pracy pomiędzy dwa dostępne zespoły. Oprogramowanie będzie powstawać 2 miesiące.

Metoda z zdefiniowanymi kontekstami:

Architekci z obu zespołów spotkali się razem i zaprojektowali kompletne rozwiązania oparte na dwóch mikroserwisach . Zdefiniowano również interfejs wymiany danych, który określił sposób wymiany danych pomiędzy dwoma serwisami, dzięki czemu obydwa zespoły mogły zacząć pracę równolegle. Zakładając, że czas potrzeby na wytworzenie jednego serwisu to 1 miesiąc powoduje, że wytwarzanie tego oprogramowania skrócił się z 2 miesięcy do 1 miesiąca.

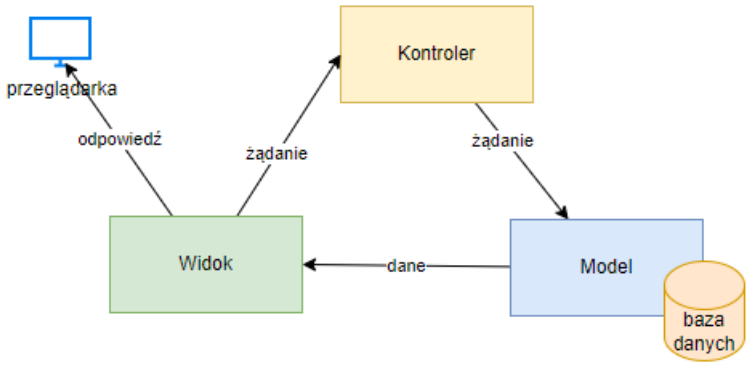
2.3 Architektura rozwiązania

System do testowania Keycloaka został stworzony w oparciu o architekturę wielowarstwową [9](rysunek 2).



Rys. 2 Architektura stworzonego rozwiązania na potrzeby testów.

Poszczególne elementy aplikacji są od siebie oddzielone i mogą być rozwijane osobno. Najczęściej występuje tutaj wzorzec projektowy MVC (ang. Model View Controller) [9], który przedstawiono na poniższym rysunku. .



Rys. 3 Model MVC.

Warstwa modelu komunikuje się z bazą danych oraz odpowiada za poprawne przetworzenie i walidacje danych. Warstwa widoku znajduje się po stronie frontendu i prezentuje dane bezpośrednio dla użytkownika systemu. Ostatnią warstwą jest kontroler, który pośredniczy pomiędzy warstwami w wymianie danych. Kontroler i model znajdują się po stronie serwera, natomiast widok jest osobną aplikacją napisaną w frameworku VueJs. Dzięki takiemu podziałowi istnieje możliwość niemalże równoległego wytwarzania frontendu i backendu

Wzorzec ten nie wspomina nic o warstwie bezpieczeństwa oraz jak zabezpieczyć aplikację. Dobrą praktyką jest zabezpieczenie warstwy widoku, ponieważ jest to miejsce, które prezentuje dane dla użytkownika. Użytkownik może nie mieć dostępu do niektórych zasobów. Więc dobrym podejściem jest ukrywanie tych widoków przed użytkownikiem. Obowiązkowo należy zabezpieczyć warstwę kontrolera, ponieważ warstwa widoku jest przetwarzana w przeglądarce użytkownika i może on nią manipulować. Backend powinien się bronić przed próbą uzyskania danych, do których użytkownik nie ma dostępu.

2.4 Rodzaje podatności

Backend

Brute force [10] – jest to atak mający na celu złamanie hasła użytkownika i uzyskanie dostępu do jego konta. Atak polega na losowym wprowadzaniu ciągów znaków w celu złamania hasła. Metoda ta jest bardzo efektywna dla krótkich haseł, ponieważ atakujący ma małą liczbę kombinacji do sprawdzenia. Również, jeżeli hasło składa się z samych cyfr lub tylko małych liter liczba kombinacji do sprawdzenia nie jest duża. Zazwyczaj złamanie takiego hasła będzie wymagało poświęcenia kilku sekund (tabela 1).

Tabela 1: Czas potrzebny do złamania hasła z określoną liczbą znaków(Opracowano na podstawie [11])

Ilość znaków	Cyfry	Małe litery	Małe + wielkie litery	Liczby, małe + wielkie litery	Liczby, symbole, małe + wielkie litery
4	Natychmiast	Natychmiast	Natychmiast	Natychmiast	Natychmiast
5	Natychmiast	Natychmiast	Natychmiast	Natychmiast	Natychmiast
6	Natychmiast	Natychmiast	Natychmiast	Natychmiast	Natychmiast
7	Natychmiast	Natychmiast	2 sekundy	7 sekund	31 sekund
8	Natychmiast	Natychmiast	2 minuty	7 minut	39 minut
9	Natychmiast	10 sekund	1 godzina	7 godzin	2 dni
10	Natychmiast	4 minuty	3 dni	3 tygodnie	5 miesięcy
11	Natychmiast	2 godziny	5 miesięcy	3 lata	34 lat
12	2 sekundy	2 dni	24 lat	200 lat	3000 lat

Inną odmianą ataku brute force jest używanie słowników najbardziej popularnych haseł. Metoda ta nie wymaga każdorazowo modyfikacji jednego znaku i próby zalogowania się na konto. Użytkownicy Internetu zazwyczaj tworzą bardzo proste hasła. Nord Pass [12] w 2022 roku stworzył listę 200 najczęściej używanych haseł. Zaskakujące jest, że na pozycji pierwszej, jest to hasło *password*, natomiast na pozycji drugiej *123456*. Skoro spora część kont jest zabezpieczona bardzo prostymi hasłami, atakujący może zebrać w formie słownika najbardziej popularne hasła i dokonać ataku. Dlatego bardzo ważne jest, żeby hasła były długie oraz zawierały znaki specjalne, cyfry, małe i wielkie litery. Innym rodzajem ataku jest odwrócony atak brute force. Polega on na próbie znalezienia poprawnego loginu. Atakujący w pliku zbiera kilka najbardziej popularnych haseł i stara się dopasować do hasła login użytkownika. Atak ten nie jest spersonalizowany, oznacza to, że atakujący nie chce włamać się na konkretne konto użytkownika, ale na dowolne konto.

Użytkownik w celu utrudnienia włamania do konta, oprócz odpowiedniej długości i różnorodności hasła, może skonfigurować mechanizm wielopoziomowego uwierzytelnienia [13], jeżeli system go wspiera. Mechanizm ten jak sama nazwa wskazuje, dodaje dodatkową warstwę bezpieczeństwa. Poza podaniem odpowiedniego hasła użytkownik będzie musiał wprowadzić drugie hasło, którym zazwyczaj jest sześciocyfrowy kod w aplikacji mobilnej. Aplikacja jest zsynchronizowana z kontem użytkownika i co 30 sekund generuje nowy kod. Przepisując kod z aplikacji uwierzytelniającej użytkownik otrzyma dostęp do konta, tylko w przypadku, jeżeli kod wygenerowany przez aplikacje, jest taki sam jak kod wygenerowany przez system. Obecnie wiele rozwiązań korzysta z bardziej rozwiniętej wersji tego systemu. Polega ona na wysłaniu notyfikacji do użytkownika, że ktoś próbuje się zalogować na konto i czy użytkownik zezwala na dokończenie procesu logowania. Rozwiązanie to jest bardziej bezpieczne, ponieważ użytkownik dostanie informacje o próbie logowania musi zdecydować czy przyznać dostęp do konta przez aplikację uwierzytelniającą, czy też nie. W przypadku

wcześniejszego rozwiązania atakujący metodą brute force, przy odrobinie szczęścia mógłby złamać ten kod. Tutaj natomiast użytkownik zawsze musi zezwolić na zalogowanie. Minusem tego rozwiązania jest dodatkowy wysiłek, który użytkownik musi podjąć w celu zalogowania się na konto. Również w przypadku utraty dostępu do aplikacji nie będzie możliwe zalogowanie się do systemu bez pomocy administratora.

SQL injection [10] – jest to podatność, która występuje najczęściej na stronach internetowych. Polega ona na wprowadzeniu zapytania sql w dostępy parametr. Zapytanie to ma za zadanie ominięcie walidacji i zwrócenie danych do atakującego. Podatność ta jest bardzo niebezpieczna, ponieważ bardzo wrażliwe dane mogą zostać wyciągnięte z systemu. Istnieje również szansa, że w przypadku braku odpowiednich zabezpieczeń po stronie serwera sql. Atakujący może zmodyfikować rekordy w bazie danych a nawet usunąć całą bazę danych. Usunięcie to spowoduje wyłączenie systemu i brak możliwości z jego korzystania. Atak SQL injection może narazić firmę na poważne straty wizerunkowe i finansowe (jak każdy inny atak – jest to truizm, a takich unikamy, aby tekst nie był zbyt rozwlekły). W celu ochrony przed tą podatnością nigdy nie należy używać bezpośrednio wartości przesłanych przez użytkownika systemu w zwykłych zapytaniach sql. Należy w tym celu użyć gotowych rozwiązań, które są dostępne na rynku. Jednym z takich rozwiązań jest Spring Data JPA [14]. Pozwala ono na przygotowywanie zapytań do bazy danych, które w pełni chronią przed podatnością SQL injection. Framework ten został stworzony według zasady domyślnie zabezpieczony ang. (Secure by Default). Oznacza to, że programista nie musi konfigurować tego narzędzia oraz stosować specjalnej konwencji, która chroni go przed tą podatnością. Framework w pełni chroni i jest odpowiedzialny za generowanie kodu wolnego od SQL injection. Programista może się w pełni skupić na wytwarzaniu logiki biznesowej.

Najprostszym testem, który sprawdzi, czy system jest podatny na ataki SQL injection, jest wprowadzenie fragmentu SQL. System zazwyczaj zwróci error typu syntax error w przypadku, gdy parametr jest bezpośrednio używany w zapytaniu. Atakujący ma teraz ułatwioną sprawę, ponieważ wie, że system jest podatny i może zacząć eksperymentować. Najczęściej próbuje się dodać warunek 1=1, który zawsze jest prawdziwy i spowoduje wyciągnięcie wszystkich rezultatów. Kolejno atakujący spróbuje dowiedzieć się jaki typ oraz wersja bazy jest używana. Dzięki tym informacją będzie w stanie znaleźć typy ataków, których może dokonać. W starszych wersjach baz danych istniała możliwość dostania się bezpośrednio na serwer i przejęcia nad nim kontroli. Dlatego dobrym nawykiem jest aktualizowanie używanych bibliotek i frameworków. Wcześniejszy sposób testowania jest jednak bardzo czasochłonny i nieefektywny. Na rynku istnieją narzędzia, które w sposób automatyczny wykonują zestaw testów pod kontem podatności na wstrzykiwanie kodu SQL. Jednym z takich narzędzi jest SQLmap opisany w dalszej części pracy.

IP Spoofing [10] - polega na podszywaniu się po oficjalne strony internetowe i organizacje. Atakujący wysyła spreparowane pakiety i fałszywe adresy IP, których nie jest właścicielem. Atakowany użytkownik nie jest w stanie wykryć tego ataku, ponieważ jest przekonany, że korzysta z oficjalnego systemu. Fałszywe adresy IP uniemożliwiają

identyfikację atakującego, przez co ta metoda jest często używana podczas przesyłania zainfekowanych maili.

Man in the middle [10] – jest to rodzaj ataku w którym atakujący pośredniczy w komunikacji pomiędzy użytkownikiem i serwerem. Cały ruch zanim dotrze do serwera jest przesyłany przez komputer atakującego. Haker może podejrzeć dane, które nie są zabezpieczone. Przechwycone dane można również zmodyfikować i wysłać dalej do serwera. Serwer zaakceptuje przesłane dane, jeżeli nie ma mechanizmu(certyfikatu), który potwierdzi zgodność danych. Komunikację można przechwycić przy pomocy ataku IP spoofing.

Frontend

XSS [10] **–** cross-site scripting jest to podatność podobna do SQL injection. Różnica polega na wstrzyknięciu do aplikacji internetowej kodu Java Script [15]. Atak ma za zadnie wykonać zainfekowany kod w przeglądarce ofiary. Przeglądarka atakowanego nie jest w stanie określić, że kod nie pochodzi z zaufanego źródła i nie powinna go wykonywać. Zainfekowany skrypt może uzyskać dostęp do ciasteczek, tokenów sesji lub innych poufnych informacji. Kod ten może podmienić zawartość strony np. strony banku. Użytkownik nieświadom tego może podać dane logowania do swojego konta bankowego i wtedy atakujący zdobędzie dane logowania. Istnieje również możliwość, że złośliwy kod będzie zdolny do wykonania przelewów pieniężnych na konto hakera. Spreparowany kod może również instalować niechciane oprogramowanie na urządzeniu ofiary. Podatność typu XSS jest bardzo niebezpieczne i należy traktować ją priorytetowo podczas wytwarzania aplikacji internetowych. Zainfekowany kod może zostać przesłany przez dowolny formularz, który nie koduje przesłanego tekstu. Przykładowo jest forum na którym każdy zalogowany użytkownik może dodawać posty. Atakujący może utworzyć post który zawiera zainfekowany kod. Przeglądarka użytkownika wczyta stronę internetową, kolejno spróbuje pobrać posty z bazy danych i wyświetlić je. Jeden post jest zainfekowany. Przeglądarka po napotkaniu tego postu wykona zawarty w nim kod. Nieświadomy w ten sposób użytkownik zostanie zaatakowany. Na szczęście współczesne frameworki domyślnie chronią przed tą podatnością. Wszelki kontent wprowadzany przez użytkownika jest kodowany, to oznacza, że nie jest bezpośrednio przetwarzany przez przeglądarkę. Często atakujący przesyła kod, który ma być wykonany w przeglądarce ofiary. Kod ten zawiera tag HTML `<script>` przeglądarka napotykając ten fragment zinterpretuje go jako fragment skryptu do wykonania i go wykona. Framowki nie wykonują bezpośrednio przesłanego kodu tylko kodują znaki specjalne. Przykładowo znak „<” zostanie zakodowany jako `&lt` a znak „>” jako `&gt`. Przeglądarka otrzyma następujący fragment kodu „&ltscript&gt”. Kod ten nie zostanie potraktowany jako do wykonania, ponieważ konwencja jest, że powinien zawierać znaki większości i mniejszości. Podeście to w pełni zabezpieczana przed atakiem XSS.

Phising [10] – jest to metoda, która ma za zadanie wyłudzić dane od użytkownika lub zmusić go do pobrania zainfekowanego oprogramowania. Najczęściej atakujący wykorzystuje komunikację mailową lub SMS w której informuje, że w systemie są problemy i potrzebna jest reakcja użytkownika. Wiadomość zazwyczaj zawiera link do strony, która bardzo przypomina stronę banku, poczty itp. Atakujący liczy, że wprowadzone zostanę poufne dane,

dzięki czemu uzyska do nich dostęp. Innym rodzajem phisingu jest sytuacja w której oficjalny system został zainfekowany i po poprawnym logowaniu, użytkownik zostanie przekierowany na spreparowany system hakera.

CSRF [10] – cross-site request forgery jest to atak, który zmusza użytkownika do wykonania akcji w systemie bez jego zgodny. Atakujący nie musi znać loginy i hasła użytkownika żeby wykonać ten atak, ponieważ atakujący nie bierzu bezpośrednio udziału w ataku. Przesyła do użytkownika zainfekowany fragment kodu, który np. zmienia adres email na podany przez atakującego. System nie jest w stanie rozróżnić, które żądanie było wykonane z własnej woli użytkownika. Dlatego wprowadzono CSRF token, który będzie dodawany do wszystkich żądań wykonanych przez użytkownika. Pozwoli to na unikniecie sytuacji w których atakujący zainfekuje aplikacje własnym żądaniem. Żądanie to zostanie odrzucone przez serwer z powodu braku tokenu CSRF. Haker nie jest w stanie zgadnąć wartości tego tokenu.

Cookie with SameSite Attribute None [10] – jest to potencjalna podatność, która pozwala przesyłać ciasteczko pomiędzy rożnymi hostami. Oznacza to, że ciasteczko to może zostać odczytane i przesłane do innej strony. Ciasteczka, które przechowuję wrażliwe informacje nigdy nie powinny mieć ustawionego parametru SameSite na None. Pozostałe dwie opcje to Strickt, która nie pozwoli przesłać ciasteczka do innego hosta niż host główny. Lax oznacza, że nie zostanie ciasteczko to wykorzystane przy żądaniach do innych stron, ale może zostać wykorzystane podczas powrotu na stronę oryginalną, przykładowo poprzez kliknięcie odnośnika do strony.

Cookie without SameSite Attribute [10] – informuje, że ciasteczko nie ma ustawionej flagi SameSite, co powoduje, że może zostać przesłane w dowolnym żądaniu. Podatność ta może spowodować ujawnienie zawartości ciasteczka. Należy bezwzględnie ustawiać flage SameSite dla ciasteczek przechowujących wrażliwe dane.

Cookie Poisoning [10] – jest to podatność, która pozwala „zatruć" ciasteczko, czyli odczytać je i kolejno zmodyfikować. Ciasteczka, które są podatne na ten atak, mogą doprowadzić do utraty danych lub zmianę zachowania systemu.

Information Disclosure – Sensitive Information in URL [10] – informacja ta udostępnia wrażliwe dane na temat użytkownika. Zagrożenie to zostanie wykryte, jeżeli np. w adresie URL przechowywany jest login i hasło. Wszystkie odwiedzone strony przechowywane są w historii przeglądarki. Dlatego należy unikać przechowywania w parametrach URL wrażliwych danych takich jak login, hasło lub token uwierzytelniający.

Loosely Scoped Cookie [10] – jest to potencjalna podatność, która umożliwia odczytanie ciasteczka z głównej strony w podstronach. Ciasteczka „Loosly scoped" są powszechnie używane w wielkich systemach takich jak Google. Należy za tym pamiętać, że zgłoszona podatność nie koniecznie musi być niebezpieczna. Każde zgłoszenie należy rozpatrzyć indywidualnie i zastanowić się czy ciasteczko udostępnia wrażliwe dane.

Modern Web Application [10] – jest to potencjalna podatność, która informuje, że atakujący ma do czynienia z nowoczesną aplikacja internetową. Każde eksponowanie informacji może być niebezpieczne, ponieważ atakujący może je wykorzystać.

User Agent Fuzzer [10] – właściwość User Agent jest ustawiana w każdym żądaniu automatycznie przez przeglądarkę i jest przesyłana do serwera. Pole to zawiera informacje na temat typu przeglądarki, wersji systemu operacyjnego, w którym uruchomiona jest aplikacja. Dzięki tym informacją aplikacja może dostosować swój wygląd do urządzenia, na którym jest wyświetlana. Niepoprawne użycie User Agent może powodować utratę kontroli nad system, ponieważ haker w parametrze prześle zainfekowany kod.

2.5 Metody autoryzacji

Rozwój komputeryzacji sprawił, że coraz więcej danych przechowywanych jest w formie cyfrowej. Chcąc uzyskać dostęp, często jest wymagane potwierdzenie tożsamości użytkownika. Poniżej przedstawiono pięć najbardziej popularnych sposobów logowania.

Password-based authentication – jest to najstarszy i najbardziej rozpowszechniony sposób autoryzacji, który polega na podaniu loginu i hasła. Hasło jest przesyłane do serwera gdzie zostaje sprawdzone z hasłem, które jest przechowywane w bazie danych. Jeżeli hasła się zgadzają użytkownik otrzymuje dostęp do zasobu. Wiele starszych systemów używa tego sposobu określenia tożsamości użytkownika. Obecnie jest on uważany za najmniej bezpieczny, ponieważ w każdym żądaniu trzeba przesłać hasło.

Multi-factor authentication – jest to metoda, która podczas próby logowania wymaga dwóch lub więcej sposobów potwierdzenia swojej tożsamości. Obecnie większość banków implementuje to rozwiązanie. Często oprócz podania hasła użytkownik musi potwierdzić wykonywaną akcję na urządzeniu mobilnym. Metoda ta jest bardziej bezpieczna ponieważ mimo złamania hasła atakujący nie uzyska dostępu do systemu bez akceptacji akcji np. na smartfonie.

Certificate-based authentication – jest to metoda która pozwala określić tożsamość, użytkownika lub urządzenia. Polega na przyznaniu dostępu na podstawie certyfikatu który został wygenerowany przez oprogramowanie zarządzające przydzielaniem dostępu. Rozwiązanie to jest bardzo bezpieczne, dopóki użytkownik lub urządzenie w odpowiedni sposób przechowuje certyfikat. W przypadku utraty certyfikatu każdy, kto uzyska do niego dostęp będzie się mógł uwierzytelnić.

Biometric authentication – jest to proces, który pozwala uwierzytelnić użytkownika na podstawie jego indywidualnych i unikalnych cech biologicznych. Sposób ten został szeroko wprowadzony w urządzeniach mobilnych, które skanowały powierzchnie palca i decydowały o przyznaniu dostępu. Nowszym podejściem jest skanowanie położenia oczu, nosa, ust przez kamerę w urządzaniu mobilnym i przyznanie dostępu w przypadku rozpoznania twarzy.

Token-based authentication – jest to sposób, która po wprowadzeniu loginu i hasła tworzy unikalny zakodowany ciąg znaków, który jest zwracany użytkownikowi. Użytkownik systemu

każdorazowo przesyła ten token podczas żądań. System rozkodowuje go i na podstawie zawartych informacji w tokenie decyduje czy przyznać dostęp do zasobów chronionych.

2.6 Metody tworzenia bezpiecznego frontendu

Frontend jest prezentowany w przeglądarce użytkownika. Należy pamiętać, że wszystko co umieścimy w kodzie będzie widoczne i dostępne dla potencjalnego atakującego. Najbardziej niebezpieczny moment jest podczas przetwarzania danych od użytkownika. Powinno się stosować zasadę ograniczonego zaufania, która polega na traktowaniu wszystkich danych jako niebezpieczne. Obecnie większość frameworków implementuje mechanizmy, które zabezpieczać prze wstrzykiwaniem złośliwego kodu. Natomiast programista musi być świadom, że wszystkie dane które umieszcza w pamięci przeglądarki mogą zostać podejrzane. Dlatego nie należy trzymać wrażliwych danych takich jak hasła w pamięci przeglądarki. Dobrą praktyką jest nieufanie danym, które są wysyłane przez serwer. Należy zdefiniować politykę bezpieczeństwa, która określi białą listę źródeł skąd mogą pochodzić dane, które będą przetwarzane przez przeglądarkę. Przesłany skrypt przez atakującego nie zostanie wykonany ponieważ źródło nie zostanie powiązane z białą listą bezpiecznych źródeł, dlatego żądanie zostanie odrzucone. Ważne jest aby informacje o wyjątkach były bardzo ogólne. Wyjątek inforumujący, że podany użytkownik nie istnieje z pewnością będzie znaczymy udogodnieniem dla użytkownika, ponieważ będzie wiedział, że podany login jest błędny. Natomiast z punktu bezpieczeństwa jest to potencjalny sposób dla atakującego na poszukiwanie odpowiedniego loginu do system, a następnie próby złamania hasła. Kolejnym ważnym aspektem jest filtrowanie danych binarnych przesyłanych przez użytkownika. W przypadku systemu, który przetwarza obrazy, przed przystąpieniem do przesyłania danych do serwera, należy sprawdzić czy przesłany plik jest obrazem, jeżeli nie to należy go odrzucić. Następną dobrą praktyką jest ukrywanie widoków, przycisków, formularzy do których użytkownik nie ma dostępu. Tutaj z pomocą przychodzi Keycloak, który jest mechanizmem potwierdzającym tożsamość użytkownika. Frontend wykorzystując ten mechanizm może określić uprawnienia użytkownika i na ich podstawie dostosować widok aplikacji.

2.7 Metody tworzenia bezpiecznego backendu

Zabezpieczenie komunikacji pomiędzy aplikacjami webowymi a serwerem jest bardzo istotne podczas wytwarzania nowoczesnych systemów. Pozwala chronić przed nieautoryzowanym dostępem do wrażliwych danych, które mogą zostać poddane modyfikacji lub usunięte. Wymiana danych pomiędzy aplikacją webową a serwerem powinna być szyfrowana. Należy użyć protokołu HTTPS, który jest szyfrowany. Atakujący, który przechwyci komunikację pomiędzy serwerem a użytkownikiem, nie będzie w stanie podejrzeć przesłanych danych, ponieważ połączenie jest szyfrowany przy pomocy klucza publicznego i tylko serwer który zna klucz prywatny może rozkodować przesłaną wiadomość. Dobrą praktyką jest aktywowanie mechanizmu uwierzytelnienia i domyślnie zabezpieczenie wszystkich punktów wejścia, wyjścia do backendu. Pozwoli to na wyeliminowanie potencjalnych miejsc w oprogramowaniu, które nie będą chronione, a powinny być.

Konfigurując mechanizm autoryzacji należy skupić się na ustawieniu odpowiedniej polityki haseł. Najlepiej jeżeli system wspiera możliwość wielopoziomowego uwierzytelniania się. Należy aktywować system wielopoziomowego uwierzytelniania się jeżeli mechanizm ten jest wspierany, ponieważ w przypadku wycieku haseł zasoby użytkowników są nadal bezpieczne. Ostatnim krokiem odnośnie zabezpieczenia API jest zdefiniowanie odpowiednich ról, które określają zasady dostępu do zasobów. Należy zdefiniować minimalny zestaw uprawnień, w celu zapobiegnięcia przypadkowego dostępu do nieuprawnionych zasobów. Podobnie jak w przypadku frontendu również na backend ważna jest walidacja przesyłanych danych i nieufanie zawartości danych, które zostały przesłane. Pozwoli to uchronić przed atakami typu SQL injection jeżeli backend ma połączenie do bazy danych. Należy używać oficjalnych bibliotek dostępowych do baz danych. Podejście to chroni przed wstrzykiwaniem złośliwych zapytań SQL. Ostatnią dobrą praktyką jest konfiguracja narzędzi, które będą analizować znalezione podatności w bibliotekach. Następnie trzeba sprawdzić, czy zgłoszona podatność jest rzeczywiście niebezpieczna. Kolejnym krokiem jest sprawdzenie, czy jest nowsza wersja biblioteki, która niweluje tą podatność. Duże systemu wymagają specjalistycznych narzędzi do logowania aktywności, które pozwalają szybko wykryć podejrzaną aktywność i poinformować odpowiednie osoby, które będą mogły zareagować na czas.

2.8 Metody autoryzacji JWT i Session

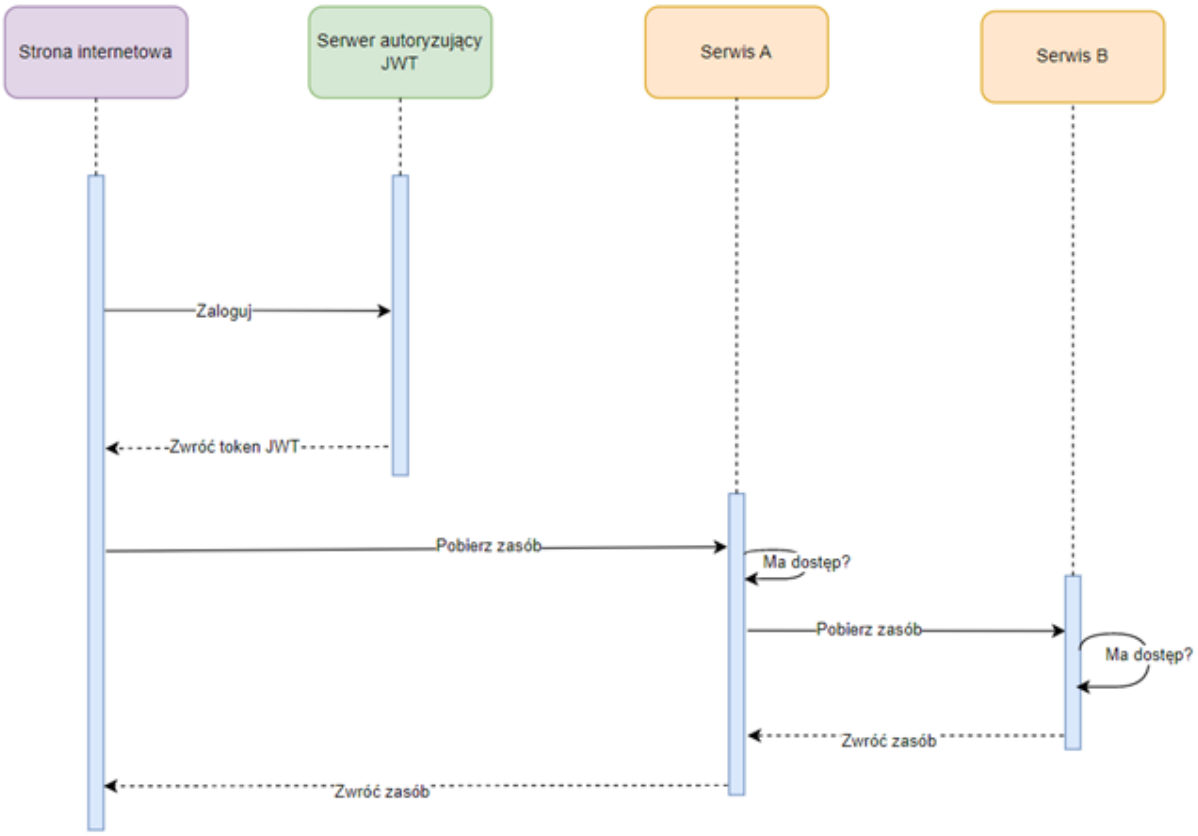
JWT (JSON web token) jest to mechanizm, który określa właściciela danych. Token (*Listing 2*) jest generowany przez serwis uwierzytelniający po podaniu odpowiedniego loginu i hasła.

Listing 2 Token JWT wygenerowany przez Keycloak podczas logowania

```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJHNmhKM25NeDNPbGhIMGxQSW1wNzdHLWh5SkdzaXNQ
OXFtWWhWSXpEbmJVIn0.eyJleHAiOjE2NzUyNzkwMDUsImlhdCI6MTY3NTI3NzI2NSwianRpIjoiY2ZiOThmNjUtNmQ0
Yy00ZDNjLThkNzktYjI2MWNjNDBiYjY0IiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgyL3JlYWxtcy9nZG1hIiwi
YXVkIjoiYWNjb3VudCIsInN1YiI6IjgyMzU5ZjAyLTM2ZjktNDE5ZC1hYmNhLTY4MTczNWE4ZDA2MCIsInR5cCI6IkJl
YXJlciIsImF6cCI6ImZyb250LWVuZC1sb2NhbGhvc3QiLCJzZXNzaW9uX3N0YXRlIjoiZmZjNmJhZmMtNzM5ZS00YWNh
LWJiYzEtN2Y0Njg4MDA0ODU2IiwiYWNyIjoiMSIsImFsbG93ZWQtb3JpZ2lucyI6WyJodHRwOi8vbG9jYWxob3N0OjUx
NzMiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbInN0dWRlbnQiLCJvZmZsaW5lX2FjY2VzcyIsImRlZmF1bHQtcm9s
ZXMtZ2RtYSIsInVtYV9hdXRob3JpemF0aW9uIiwidXNlciJdfSwicmVzb3VyY2VfYWNjZXNzIjp7ImFjY291bnQiOnsi
cm9sZXMiOlsibWFuYWdlLWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZpZXctcHJvZmlsZSJdfX0sInNj
b3BlIjoib3BlbmlkIHByb2ZpbGUgZW1haWwiLCJzaWQiOiJmZmM2YmFmYy03MzllLTRhY2EtYmJjMS03ZjQ2ODgwMDQ4
NTYiLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwibmFtZSI6Ik1pY2hhxYIgRmxvcmN6YWsiLCJwcmVmZXJyZWRfdXNlcm5h
bWUiOiJtaWNoYWwiLCJnaXZlbl9uYW1lIjoiTWljaGFgiIsImZhbWlseV9uYW1lIjoiRmxvcmN6YWsiLCJlbWFpbCI6
Im1pY2hhbC5mbG9yY3phazBAZ21haWwuY29tIn0.QmiZA4uhFyV38GVuCYjeWPMITJEcziMNv5c1AtlT_xAXRQbbmEO0
aDDiRRYvrErDDfCIP317c0KxTuH94yMnhk4kIS61G3UmTouB7jw1AhX9F8-
42Ah5ZznampUwdgJ3bOw4hCDW63BLEXnYViPcML3_u--IE6xJd2mxivP4LybV0SnYtTghxCGelXCc1-RH4DGXwULl6-
6BjmXwpU2FlczAYWpaYnS6GwXOadC6ecPTom1Ali9FCgNXQZOIrE3Zy7I9tudWMX9mDWQc3oDJkRkI9hledjcgwm88LQ
7WjoZQyNdVlu3FdykbF-l6bWJoIjlNv4u8CzDeYIP-wH9AOg
```

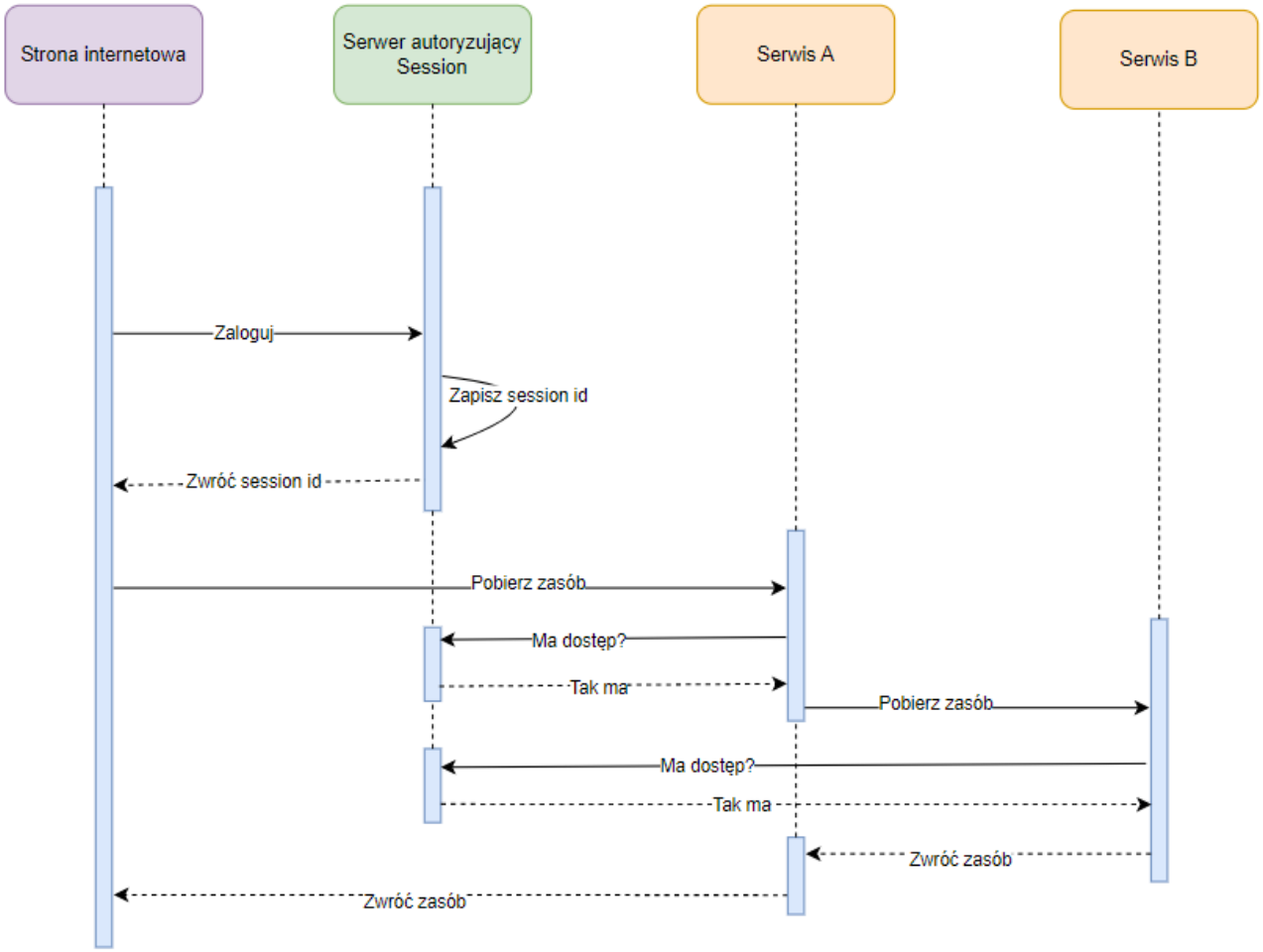
Token ten jest zaszyfrowany i podpisany kryptograficznie. Serwer po otrzymaniu i weryfikacji tokenu może zaufać użytkownikowi, ponieważ token jest podpisany cyfrowo, przez co serwis, który otrzymał token jest w stanie sprawdzić, czy token nie został zmodyfikowany. Man in the middle nie może zmodyfikować tego tokenu, ponieważ jeżeli to zrobi aplikacja odrzuci taki token i zablokuje dostęp, ponieważ część zaszyfrowana nie będzie się zgadzać z danymi niezaszyfrowanymi. . JWT jest przesyłany w formie zaszyfrowanych danych algorytmem base64. Algorytm ten zamienia dane z formy binarnej na ciąg znaków

tekstowych. Przez to, że token jest przesyłany w formie tekstowej tzn. atakujący w łatwy sposób może rozszyfrować token, dlatego nie powinno się wewnątrz tego tokenu przechowywać danych innych niż role, imię, email. . Problemem tokenów JWT jest niemożliwość anulowania ważności tokenu. Przykładowo, jeżeli token jest ważny przez jeden dzień i użytkownik ma uprawnienia administratora to po redukcji tych uprawnień system nie jest w stanie zablokować takiego tokenu. Dopóki token będzie ważny użytkownik może z niego korzystać, dlatego dobrą praktyką jest tworzenie tokenów krótko żyjących. JWT także nie wspiera mechanizmu „wyloguj ze wszystkich urządzeń", ponieważ z punktu widzenia systemu, jeżeli użytkownik ma token to ma dostęp do zasobu. Istnieje możliwość unieważnienia prywatnego klucza, którym został podpisany token, ale spowoduje to utratę ważności wszystkich tokenów. W najgorszym przypadku wszyscy użytkownicy utracą dostęp do aplikacji, ponieważ nowy klucz nie będzie się zgadzał z starym. Skoro nie ma łatwej możliwości unieważnienia tokenu JWT, to dobrą praktyką jest przechowywanie go w ciasteczku z flagą `httpOnly` w przeglądarce użytkownika. Flaga ta zapewnia, że to ciasteczko będzie wysyłane tylko do żądań HTTP do serwera. Pozwoli to uniknąć sytuacji, w których złośliwe oprogramowanie będzie w stanie odczytać zawartość ciasteczka z przeglądarki. Nie powinno się przechowywać tego tokenu w parametrze URL, ponieważ zostanie on w historii przeglądarki. Przeglądarka posiada pamięc podręczną (`localStorage`), w którym można przechowywać dane. Każdy skrypt wykonywany w przeglądarce ma dostęp do tej pamięci. Dlatego złym pomysłem jest przechowywanie tego tokenu w pamięci podręcznej. Mechanizm JWT jest również bardziej efektywny, ponieważ po otrzymaniu tokenu serwis jest sam w stanie sprawdzić, czy jest poprawny i czy użytkownik ma dostęp do zasobu (rysunek 4).



Rys. 4 Diagram sekwencji autoryzacji do zasobu dla JWT

Sesja jest tworzony w przypadku jeżeli użytkownik przesłał poprawny login i hasło. Najczęściej po udanym logowaniu w bazie danych jest tworzony rekord, który zawiera takie informacje jak unikalny identyfikator sesji, czas logowania, okres ważności sesji. Poprawne zalogowanie powoduje zwrócenie do użytkownika unikalnego ciągłu znaków, którym jest identyfikator sesji. Identyfikator sesji jest najczęściej przechowywany w ciasteczku in jest używany przy pobieraniu zasobów z serwisów. Serwis, który otrzyma id sesji, wyśle żądanie do serwera autoryzującego, w celu sprawdzenia poprawności identyfikatora sesji. Rozwiązanie to jest bardzo bezpieczne, ponieważ istnieje tylko jedno miejsce, w systemie które przetrzymuje i sprawdza poprawność uprawnień użytkownika. Nie istnieje tutaj problem unieważnia sesji użytkownika jak w przypadku JWT. Chcąc wylogować użytkownika z wszystkich urządzeń wystarczy usunąć rekord z informacją o sesji w bazie danych serwera autoryzującego. Minusem tego rozwiązania jest konieczność każdorazowo wykonywania zapytania do serwera w celu sprawdzenia, czy przesłane id sesji jest poprawne. W przypadku dużej ilości serwisów biorących udział w przetwarzaniu danych, możliwy będzie znaczny spadek wydajności, ponieważ każdy serwis będzie wysyłał żądania w celu sprawdzenia uprawnień (rysunek 5).



Rys. 5: Diagram sekwencji autoryzacji do zasobu dla Session

2.9 Przegląd narzędzi do testowania

Wireshark [16] – jest to narzędzie, które umożliwia poddaniu analizie pakietów sieciowych. Istnieje możliwość podglądania zawartości pakietów na żywo lub analizy historycznych pakietów które zostały zapisane na serwerze. Narzędzie jest darmowe i może być używane na licencji open source. Najczęściej jest używany przez administratorów systemu w celu znalezienia miejsc w systemie, które nie działają tak jak oczekiwano. Inżynierowie cyberbezpieczeństwa wykorzystują to narzędzie do śledzenia zawartości przesyłanych żądań i w przypadku ataków analizy zawartości pakietów, w celu znalezienia miejsc w systemie, które potencjalnie zostały uszkodzone.

OWASP ZAP [8]– jest to narzędzie, które testuje bezpieczeństwo aplikacji oraz stron internetowych. Oprogramowanie jest typu open source i jest rozwijane przez międzynarodową grupę wolontariuszy. ZAP może być uruchomiony w dwóch trybach. Jednym z nich jest automatyczny skan aplikacji. Tester podaje adres URL testowanej aplikacji i uruchamia zadanie, które dokonuje testów penetracyjnych. Testy te sprawdzają możliwe zasoby, gdzie mogą się dostać i próbują dokonać znanych ataków np. SQL injection. Znaleziona podatność jest natychmiast raportowana w interfejsie. Znajduje się tam krótki opis

wykrytej podatności oraz jeżeli jest to możliwe, to oprogramowanie sugeruje sposób zabezpieczenia się. Aplikacja nie jest w stanie ocenić czy wykryte zagrożenie jest niebezpieczne, dlatego tester musi sam ocenić, czy aplikacja jest zagrożona czy nie. Systemy automatyczne nie są doskonałe, ponieważ nie znają kontekstu przeprowadzonego testu, dlatego użytkownik zawsze musi samemu zweryfikować i zinterpretować wynik testów. Kolejną funkcją, którą udostępnia narzędzie ZAP jest możliwość skonfigurowania proxy, które przekieruje cały ruch do skanera. Użytkownik konfiguruje aby cały ruch z określonego portu w przeglądarce został przekierowany do narzędzia ZAP. Następnie zamiast wykonywać automatycznych testów. Tester manualnie otwiera aplikacje internetową i ręcznie odwiedza różne podstrony. Cały ruch idzie jednocześnie przez ZAP, który skanuje przesłane zawartości i próbuje znaleźć podatności. Zaletą tego podejścia jest fakt, że wszystkie funkcje systemu mogą zostać przetestowane i żadna nie zostanie pominięta. Minusem natomiast jest czasochłonność tego sposobu, ponieważ są to testy półautomatyczne. ZAP posiada narzędzie „Fuzzer", które umożliwia wysyłanie spreparowanych ładunków do aplikacji. Ładunki można przygotować samemu lub użyć gotowych przegotowanych przez społeczność aplikacji. Należy jednak mieć na uwadze, że podczas testów można uszkodzić dane, aplikacje lub cały serwer. Dlatego testów należy dokonywać na specjalnych instancjach przygotowanych specjalnie do testów penetracyjnych. Nigdy nie należy wywoływać tych testów na instancjach produkcyjnych.

Burp Suite [17] – jest to narzędzie, które pozwala testować bezpieczeństwo aplikacji. Przeciwnie do poprzedniego rozwiązania tylko wersja „społecznościowa" jest darmowa. Nie zawiera ona żadnych automatycznych testów. Tester musi manualnie testować aplikację, poprzez przechwytywanie żądań, analizę ich i ewentualną modyfikację w celu znalezienia podatności. Burp posiada mechanizm Spider, który potrafi zbierać adresy do zasobów jakie istnieją w aplikacji internetowej. Więcej zebranych adresów może posłużyć do przygotowania większej ilości ataków. Często jako autor oprogramowania, może być nie świadomy, że eksponuje na świat zasoby, które powinny być prywatne. Z pomocą tej funkcjonalności można to zweryfikować. Kolejną funkcjonalnością jest możliwość przechwytywania i modyfikacji żądań. Skonfigurowanie odpowiedniego proxy pozwala podejrzeć i modyfikować zawartości ładunków. Przechwycone ładunki zapytań mogą zostać zapisane do pliku i wykorzystane we własnych testach penetracyjnych lub dedykowanych rozwiązaniach, które testują aplikację. Wersja enterprise pozwala skonfigurować skaner, który regularnie skanuje całą aplikację w celu znalezienia luk w systemie. Wszelkie znalezione podatności są raportowane do administratora systemu.

SQLmap [10]– jest to narzędzie napisane w języku Python [18], które służy to ataków sql injection w celu wyciągnięcia wrażliwych danych lub uzyskania dostępu do systemu. Narzędzie jest typu open source. Przed przeprowadzeniem ataku biblioteka, dokonuje analizy, która ma za zadanie rozpoznać bazę danych której używa system. Pozwoli to na przygotowanie najbardziej dopasowanych testów penetracyjnych, które spróbują włamać się do systemu. Narzędzie nie tylko pozwala podejrzeć dane w bazie danych, ale również w przypadku słabego zabezpieczenia serwera istnieje szansa przejęcia kontroli nad nim.

Biblioteka po udanym ataku i uzyskaniu dostępu do bazy danych pozwala zrzucić dane z tabel na komputer atakującego. Rozwiązanie to jest w pełni automatyczne i nie wymaga od atakującego żadnej specjalistycznej wiedzy. Domyślnie biblioteka próbuje dostać się na serwer wstrzykując sql w dostępne parametry, po udanym ataku i dostaniu się na serwer. Atakujący może nawet w niektórych przypadkach uzyskać dostęp do systemu operacyjnego.

Postman [19] – jest to narzędzie, które pozwala wykonywać żądania HTTP i sprawdzać ich rezultat. Wykorzystywane jest podczas testowania poprawności połączenia z backendem. Wspiera wszystkie popularne mechanizmy autoryzacji, przez co może wykonać każde żądanie do serwera. Istnieje możliwość tworzenia testów, które będą walidować, czy przesłana odpowiedź z serwera pokrywa się z oczekiwaną odpowiedzią.

II Część praktyczna

3. Implementacja

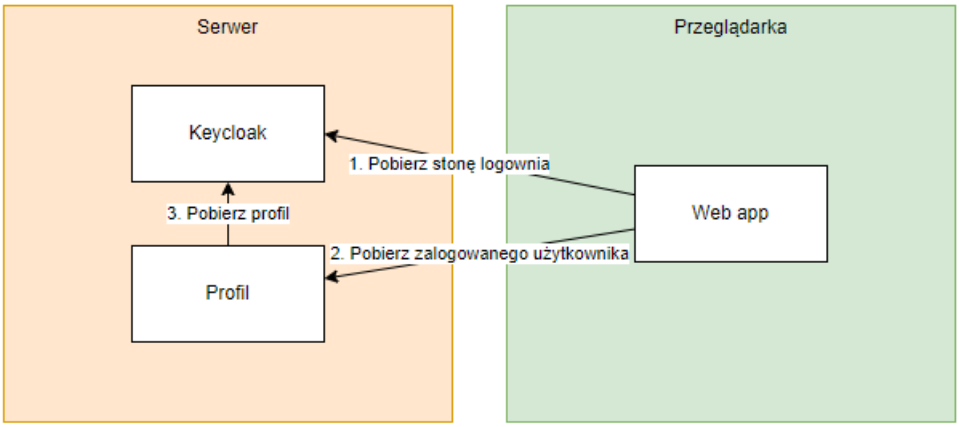
Część ta prezentuje proces zbierania wymagań, konfigurację infrastruktury oraz przygotowanie aplikacji do przeprowadzenia testów penetracyjnych. Kolejno przedstawia wyniki testów oraz zalecane konfiguracje zwiększające bezpieczeństwo w testowanej aplikacji.

3.1 Zbieranie wymagań

Rozwiązanie przede wszystkim musi być bezpieczne, ponieważ zostanie wykorzystane w startupie. Dlatego bezpieczeństwo to najważniejsze wymaganie, ponieważ będzie miało znaczący wpływ na sukces startupu. System, które będzie podatny na ataki nie będzie szeroko używany przez użytkowników. Kolejnym wymaganiem jest, że oprogramowanie uwierzytelniające musi być darmowe, ponieważ startup nie posiada dużego budżetu pieniędzy, dlatego ważne jest szukanie miejsc, w których można optymalizować koszty. Ważne jest, aby mieć pełną kontrole nad mechanizmem uwierzytelniającym, żeby mógł być zainstalowany i zarządzany na własnym serwerze. Następnym wymaganiem jest, żeby mechanizm autoryzacji dostarczał interfejs użytkownika, który jest w pełni modyfikowalny. Biorąc pod uwagę wszystkie wymagania do testów wybrano Keycloka, ponieważ pokrywa większość wymagań. W sieci nie znaleziono informacji na temat bezpieczeństwa rozwiązania Keycloak, dlatego postawiono w tej pracy to zbadać.

3.2 Przygotowanie infrastruktury

Zanim przystąpiono do implementacji zaprojektowano środowisko (rysunek 6), na którym będą dokonywane testy. Podjęto decyzję, że Keycloak zostanie skonfigurowany zarówno dla backendu jak i frontendu. Pozwoli to na pełne poznanie możliwości tego narzędzia jak i nabranie wprawy w posługiwaniu się nim.



Rys. 6 Projekt infrastruktury

Zdecydowano, że zostanie stworzony serwis o nazwie Profil, który będzie odpowiadał za zarządzanie danymi użytkownika, takimi jak imię, nazwisko, email. Aplikacja webowa, po poprawnym uwierzytelnieniu otrzyma JWT z Keycloka. Token ten zostanie przesłany do serwisu Profil w celu pobrania imienia zalogowanego użytkownika. Dokona tego poprzez wywołanie żądania do serwisu Profil. Serwis ten na podstawie identyfikatora użytkownika, który znajduje się w tokenie, spróbuje pobrać użytkownika z bazy danych. Sytuacja w której użytkownik nie znajduje się w bazie danych wymaga żądania do Keycloak w celu pobrania danych na temat zalogowane użytkownika. Pobrane dane zostaną zapisane w bazie danych serwisu Profil i zwrócone do aplikacji webowe. Każde kolejne żądanie o ten sam profil będzie powodowało wczytanie go z bazy danych serwisu o nazwie Profil.

Keycloaka uruchomiono w domyślnym trybie dev. Baza danych jest w pamięci i komunikacja odbywa się przez protokół http, który nie jest szyfrowany. Poniżej zaprezentowano minimalną konfiguracje docker compose (*Listing 3*), która jest wymagana do uruchomienia aplikacji Keycloak.

Listing 3 Konfiguracja docker compose

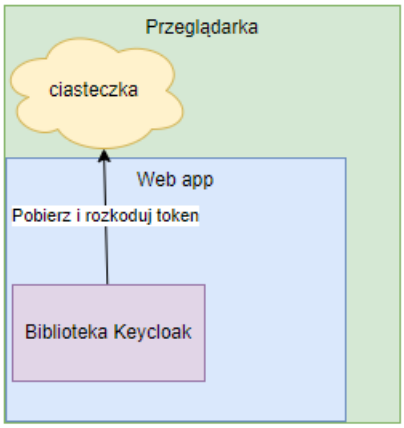
```
services:
  keycloak:
    hostname: keycloak-svc
    image: quay.io/keycloak/keycloak:21.0.1
    ports:
      - "8082:8080"
    command: start-dev  --hostname-url=http://localhost:8082
    environment:
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
```

Backend został zabezpieczony przy pomocy adaptera Keycloak. W celu przetworzenia żądania użytkownik musi wysłać w nagłówku `Authentication` token JWT . Token ten zostanie rozpakowany i porównane zostaną role użytkownika z *Listing 4*. Zgodność roli z tokenu z rolą w konfiguracji, spowoduje przyznanie użytkownikowi dostępu. Konfiguracja jest bardzo prosta. Wystarczy podać nazwę klasy zasobu oraz określić do których metod HTTP dana rola ma dostęp. Administrator może wyświetlić wszystkie profile użytkowników, może je tworzyć, edytować, przeglądać, usuwać. Natomiast zwykły użytkownik z rolą `user`, może pobrać tylko swój profil.

Listing 4 Konfiguracja dostępu do zasobu

```
@Configuration
public class ResourceSecurity implements SecurityModuleConfigurer {

    @Override
    public void configure(SecurityConfig.Builder config) {
        // Zdefiniowanie ról I zasobów, do których konkretne role mają dostęp.
        config.permitRole("admin", Profile.class, ResourcePermission.ALL);
        config.permitRole("user", CurrentProfile.class, ResourcePermission.GET);
    }
}
```

Frontend został zabezpieczony przy pomocy biblioteki dostarczonej przez Keycloak. Utworzono konfigurację jak na *Listing 5*. Konfiguracja ta jest odpowiedzialna za sprawdzanie, czy użytkownik jest zalogowany. Biblioteka z ciasteczek przeglądarki pobiera token i rozkodowuje go z formatu base64 do formatu JSON (rysunek 7).



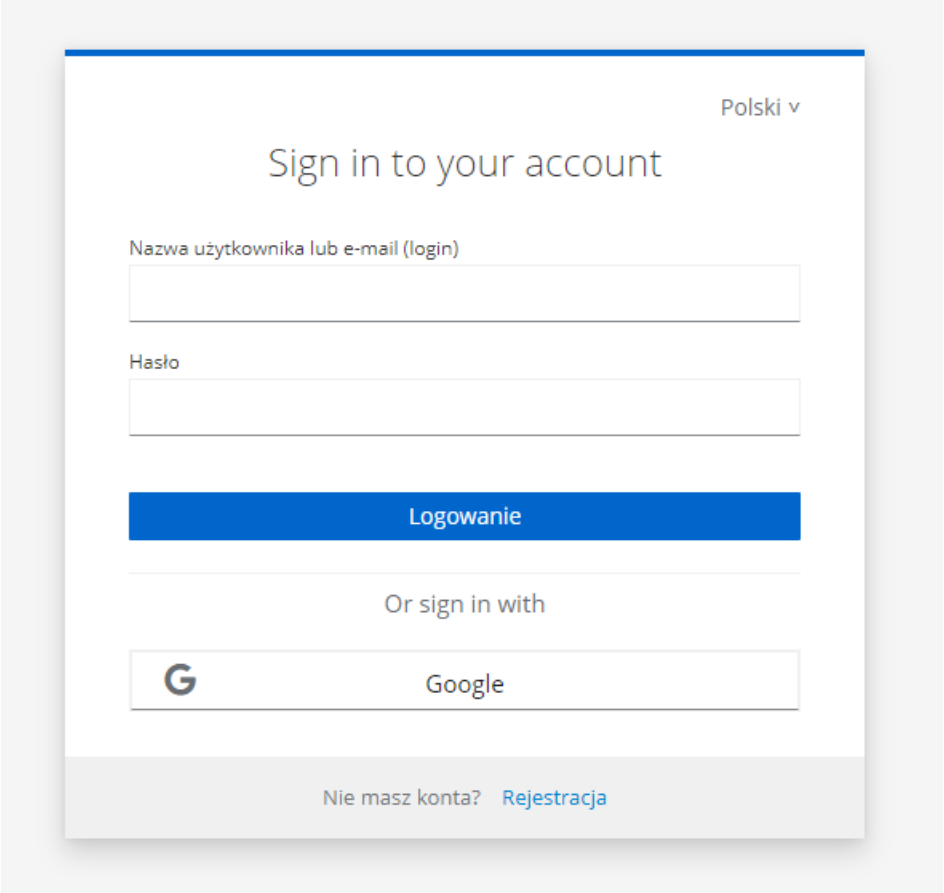
Rys. 7 Pobieranie tokenu z ciasteczka

Następnie sprawdza czy format jest poprawny oraz, czy token nie wygasł. Biblioteka ładuje aplikacje w przypadku, jeżeli token przeszedł walidacje. Użytkownik otrzyma dostęp do strony internetowej i będzie w stanie wyświetlić jej zawartość. Istnieje również możliwość sprawdzenia, czy użytkownik ma określoną role i na podstawie tej roli wyświetlić zawartość strony, do której ma dostęp. Należy jednak pamiętać, że zabezpieczenie frontendu jest bardzo ważne, ale to za mało, żeby cały ekosystem był bezpieczny. Dobrą praktyką jest ustawianie krótkiego czasu ważności tokenu. Krótko żyjący token nawet w przypadku przechwycenia przez osoby trzecie szybciej wygaśnie i nie będzie zdatny do użytku.

Listing 5 Filtr przekierowujący użytkownika na stronę logowania w frontend

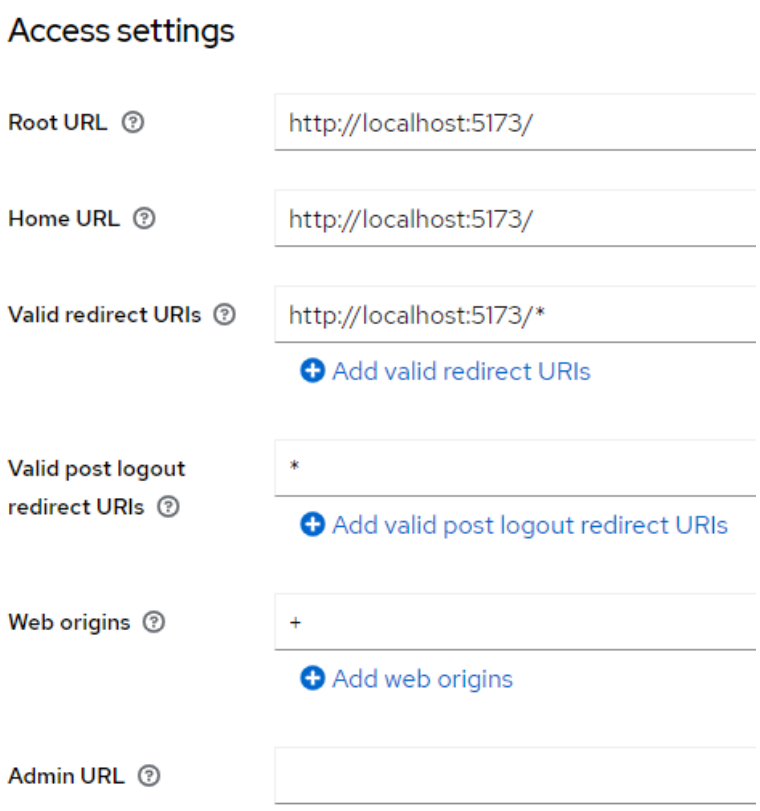
```
//Pobranie z zmiennych środowiskowych adresu URL do serwera Keycloak.

const keycloakHostname = import.meta.env.VITE_KEYCLOAK_HOST;

//Pobranie z zmiennych środowiskowych nazwy klienta Keycloak
const keycloakClient = import.meta.env.VITE_KEYCLOAK_CLIENT;


//Stworzenie obiektu przechowujacego niezbędne dane do działania.
const config: KeycloakConfig = {
    url: keycloakHostname, realm: "pk", clientId: keycloakClient
}


//Zainicjalizowano obiekt Keycloak z wcześniejszą konfiguracją.

const keycloak = new Keycloak(config);


//Skonfigurowano opcje, które będą sprawdzane przy pierwszym odwiedzeniu strony internetowej.
const initOptions: KeycloakInitOptions = {

    //Opcja ta sprawdza, czy w ciasteczku jest token, jeżeli jest użyje go. Rozwiązanie to jest
//bardzo wygodne ponieważ pozwala zalogować się raz w jednej aplikacji i reużywać tokenu w innych.
    onLoad: "check-sso",

    //Opcja ta ukrywa okno logowania przy sprawdzaniu poprawności tokenu. Gdyby była wyłączona
//użytkownik na moment zostałby przekierowany na stronę logowania.
    checkLoginIframe: true
}

keycloak.init(initOptions)
.then(isAuthorize => {
    if (isAuthorize) {

        // Jeżeli użytkownik jest zalogowany, to wyświetli główną stronę aplikacji.
        app.mount("#app");
    } else {

        //Jeżeli użytkownik nie jest zalogowany, to zostanie przekierowany na stronę logowania
//Keycloak.

      window.location.reload()
    }
    //Dobrą praktyką jest tworzenie tokenów krótko żyjących, dlatego w celu uniknięcia częstej
//potrzeby wprowadzania hasła, skonfigurowano mechanizm, który co 30 sekund wyśle, żądanie do
//Keycloak w celu wygenerowaniu nowego tokenu. Wszystko odbywa się w tle, dlatego proces ten nie
//wpływa negatywnie na komfort korzystania z aplikacji internetowej.
    setInterval(() => {
        keycloak.updateToken(70)
        .then()
        .catch(() => {
          console.error('Failed to refresh token');
        });
      }, 30000)

}).catch(console.error) [2]
```

Skonfigurowany mechanizm z *Listing 5* sprawia, że jeżeli użytkownik nie jest zalogowany zostanie przekierowany na stronę logowania (rysunek 8).



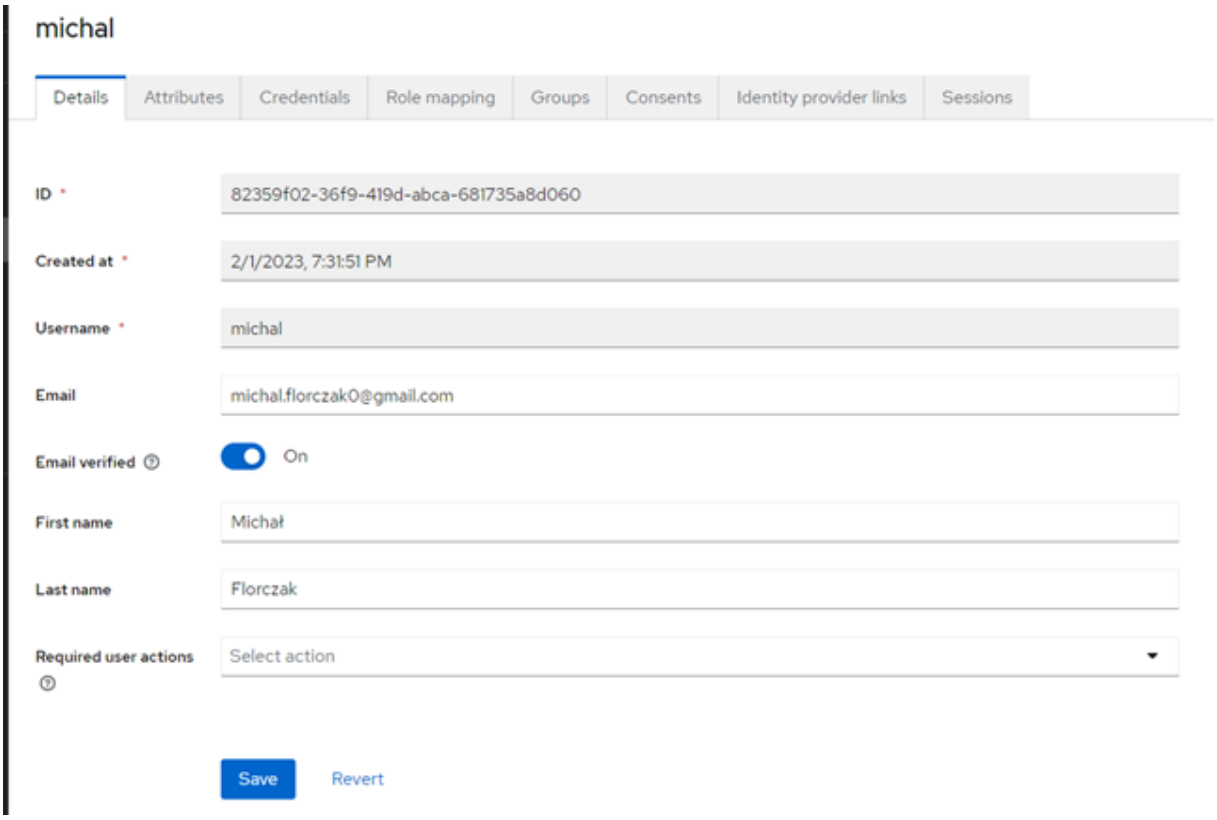
Rys. 8 Widok formularza do logowania

Kolejnym krokiem jest skonfigurowanie klienta (rysunek 9). Konfiguracja polega na poinformowaniu Keycloak na który adres url aplikacja powinna przekierować użytkownika po poprawnym jego uwierzytelnieniu. Istnieje możliwość skonfigurowania adresu na który ma zostać przekierowany widok aplikacji po wylogowaniu się. Web origins określa jakie hosty mogą używać serwera autoryzującego. Plus oznacz, że do serwera uwierzytelniającego będą miały dostęp aplikacje, które uruchomione są na tym samym hoście. Podejście to jest bardzo bezpieczne, ponieważ chroni przed wywoływaniem żądań z zewnątrz serwera.



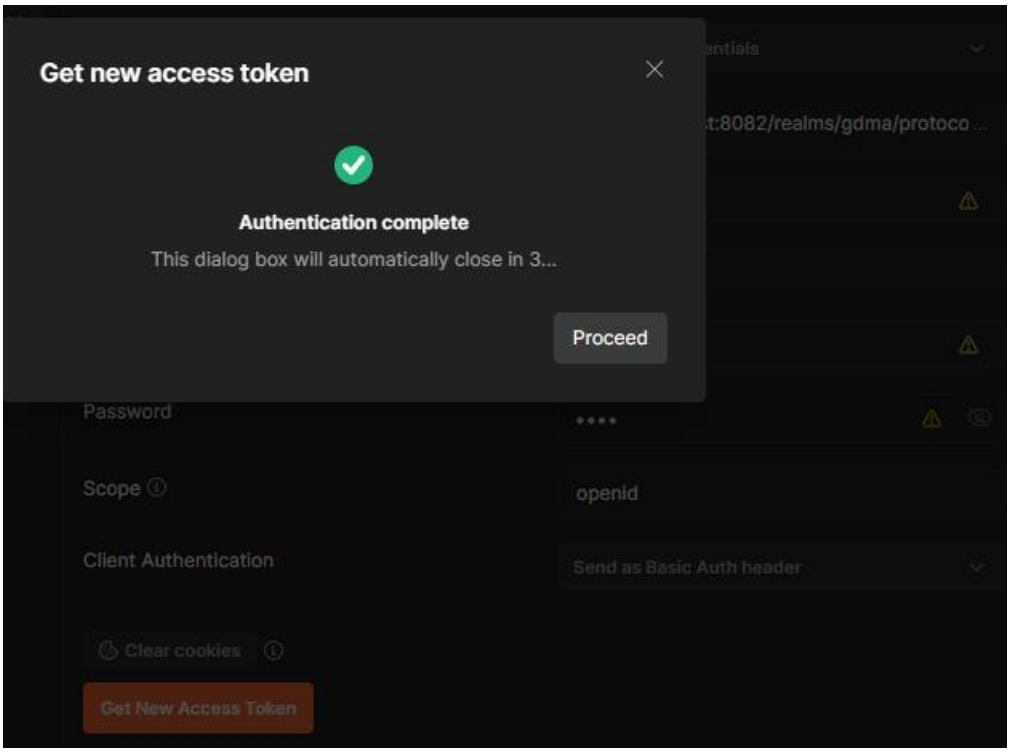
Rys. 9 Widok ustawień klienta

Kolejno stworzono nowego użytkownika, którego przypisano do nowej roli student, poniżej na rysunku 10 przedstawiono jego detale. Można zaobserwować, że każdy użytkownik posiada indywidualne Id, które jest niemodyfikowalne. Nazwa użytkownika jest niemodyfikowalna. W sytuacji błędnej nazwy użytkownika nie istnieje możliwość jej edycji. Należy takie konto usunąć i utworzyć nowe. Administrator posiada możliwość do zmuszenia użytkownika do wykonania określonej akcji po udanym logowaniu. Akcją może być np. zmiana hasła.



Rys. 10 Detale stworzonego użytkownika

Przy pomocy narzędzia Postman przetestowano możliwość logowania się na powyższe konto użytkownika w celu pobrania tokenu JWT. Ustawiono adres do serwera autoryzacji z którego można pobrać token. Kolejno wybrano formę uwierzytelnienia. Następnie po podaniu poprawnego loginu oraz hasła, wykonano żądanie. Rysunek 6 zawiera informację, że udało się pobrać token dla wprowadzonych danych logowania. Również w odpowiedzi z serwera znajduje się token, który służy do uwierzytelnienia użytkownika w systemie.



Rys. 11 Udane Logowanie na konto użytkownika

Kolejno sprawdzimy, czy w odebranym tokenie (*Listing 2*) znajdują się informacje na temat ról użytkownika. W celu weryfikacji tych danych użyta zostanie witryna jwt.io na której można rozszyfrować token i podejrzeć jego zawartość. Token z *Listing 2* po rozszyfrowaniu z formatu base64 jest w formacie JSON. *Listing 6* prezentuje jego rozszyfrowaną postać.

Listing 6 Rozszyfrowany zawartości tokenu JWT stworzony przez Keycloak

```
{
  "exp": 1675279005,
  "iat": 1675277265,
  "jti": "cfb98f65-6d4c-4d3c-8d79-b261cc40bb64",
  "iss": "http://localhost:8082/realms/gdma",
  "aud": "account",
  "sub": "82359f02-36f9-419d-abca-681735a8d060",
  "typ": "Bearer",
  "azp": "front-end-localhost",
  "session_state": "ffc6bafc-739e-4aca-bbc1-7f4688004856",
  "acr": "1",
  "allowed-origins": [
    "http://localhost:5173"
```

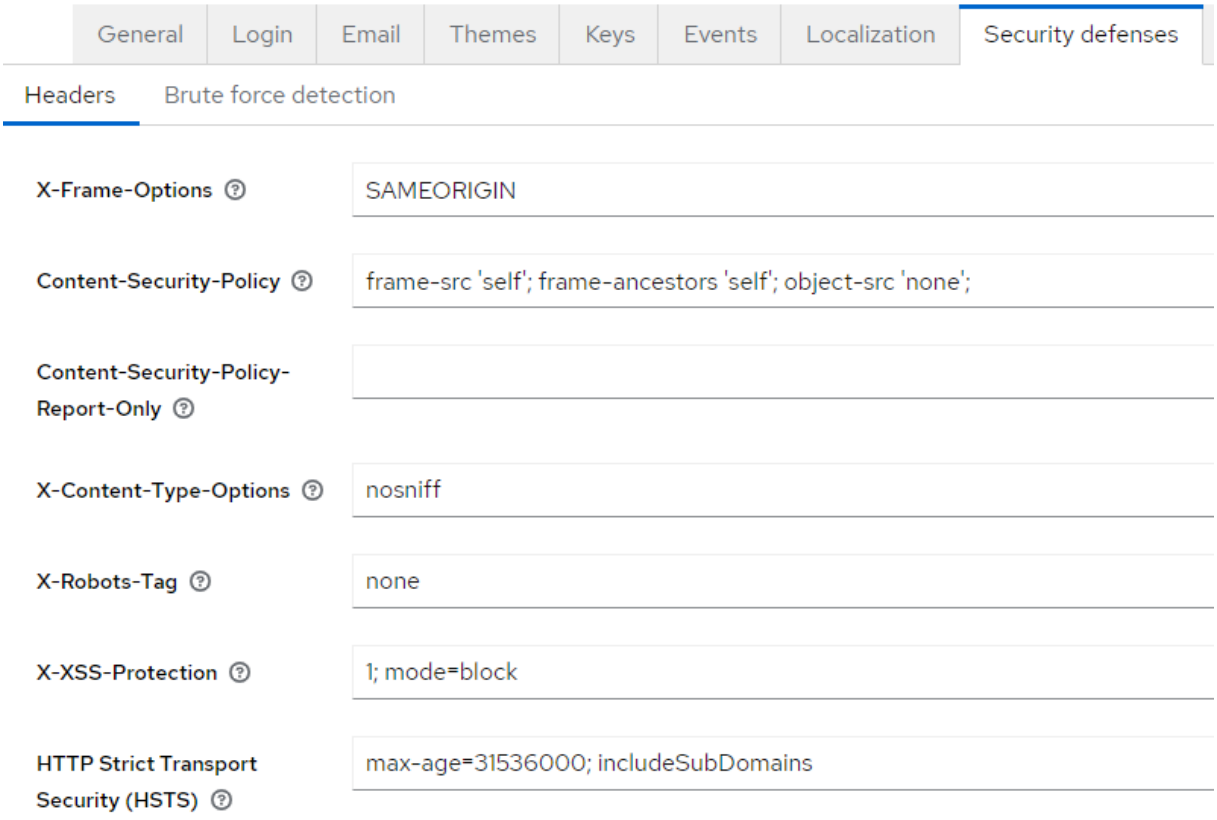
```
  ],
  "realm_access": {
    "roles": [
      "student",
      "offline_access",
      "default-roles-gdma",
      "uma_authorization",
      "user"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid profile email",
  "sid": "ffc6bafc-739e-4aca-bbc1-7f4688004856",
  "email_verified": true,
  "name": "Michał Florczak",
  "preferred_username": "michal",
  "given_name": "Michał",
  "family_name": "Florczak",
  "email": "michal.florczak0@gmail.com"
}
```


Informacje z powyższego JSON prezentują role jakie posiada użytkownik, możemy zauważyć w nich rolę „student”. Ważnymi parametrami są `exp i iat`. Pierwszy parametr określa czas wygaśnięcia tokenu. Natomiast drugi przechowuje datę i czas kiedy token został wydany. Parametry te dwa są bardzo istotne, ponieważ dzięki nim aplikacja działająca po stronie serwera jest w stanie określić czy otrzymany token jest ważny oraz czy powinna na

jego podstawie przyznać użytkownikowi dostęp do zasobów. Token zawiera również informacje o imieniu, nazwisku, adresie email. Mechanizmy bezpieczeństwa, będą wykorzystywały te informacje w celu zezwolenia lub zabronienia użytkownikowi na dostęp do zabezpieczonych zasobów serwera.

3.3 Analiza aplikacji Keycloak z domyślną konfiguracją

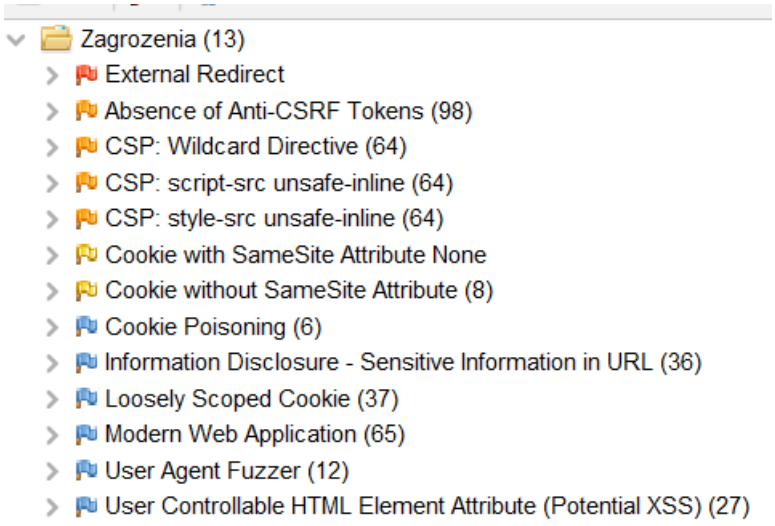
Domyślna konfiguracja bezpieczeństwa dla Keycloak znajduje się na rysunku 12. Można w niej skonfigurować obronę przed atakami typu XSS. Skonfigurować mechanizm, który będzie wymuszał na przeglądarce internetowej, komunikację zaszyfrowanym protokołem http. Domyślnie komunikacja odbywa się nieszyfrowanym protokołem.



Rys. 12 Domyślna konfiguracja bezpieczeństwa dla Keycloak

Użytkownik próbuje wejść na stronę główną stworzonej aplikacji webowej. . Sytuacja, w której użytkownik nie jest uwierzytelniony spowoduje, że zostanie przekierowany na stronę logowania pobraną z Keycloak. Największym zagrożeniem jest tutaj „External Redirect", czyli tłumacząc na język polski zewnętrzne przekierowanie. Przekierowanie to jest używane do powrotu użytkownika na stronę, którą chciał odwiedzić przed zalogowaniem się. Złe zaimplementowanie tego mechanizmu może doprowadzić do phisingu, który jest bardzo niebezpieczny dla użytkownika. Atakujący może podmienić adres przekierowania na własną stronę, która najczęściej jest imitacją strony internetowej, którą chciał odwiedzić użytkownik. Doprowadzić to może do wycieku danych wrażliwych, pobrania zainfekowanego

oprogramowania itp. Rozwiązać ten problem można poprzez blokowanie wszystkich prób przekierowań na nieznane witryny. Domyśla konfiguracja nie jest poprawna, ponieważ zawiera „*”, czyli tak zwany wildcard, który akceptuje wszystkie adresy URL. Zmiana wartości na dokładny adres witryny wyeliminuje to zagrożeniem i aplikacja nie będzie już podatna na phising.



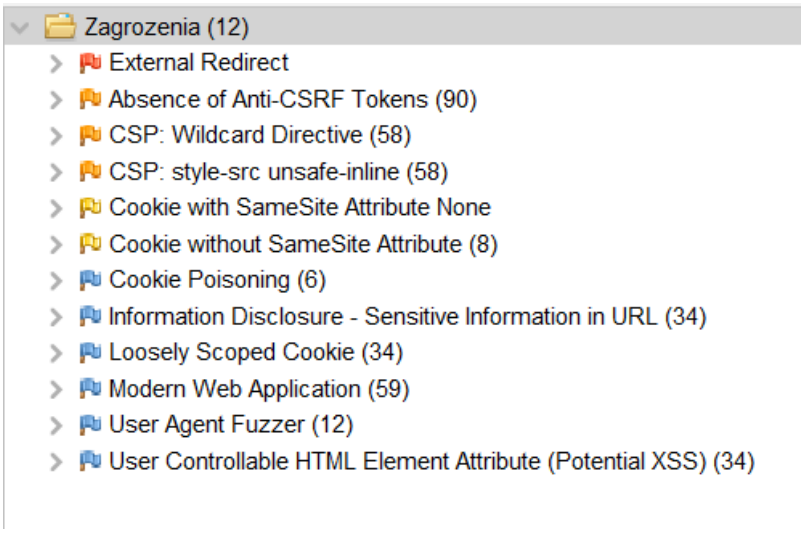
Rys. 13 Widok pierwszej analizy aplikacji

„CSP script-src unsafe-inline” problemem tutaj jest zezwolenie wczytywania wszystkich skryptów javascript. Konfiguracja ta jest niebezpieczna, ponieważ atakujący może wstrzyknąć kod javascript, który zostanie wykonany w przeglądarce atakowanej osoby. Domyślą konfigurację dla Content-Security-Policy przedstawiono na *Listing 7*.

Listing 7 Domyślna konfiguracja dla CSP script

```
frame-src 'self'; frame-ancestors 'self'; object-src 'none';
```

Brakuje powyżej `script-src 'self'`, która sprawia, że tylko skrypty pochodzące z tego samego serwera będą wykonywane. Dodanie brakującego elementu zmniejsza ilość zagrożeń. Rysunek 14 przedstawia obecny stan zagrożeń dla aplikacji. Kolejne zagrożenia będą wyeliminowywane w analogiczny sposób. Zamiast używać domyślnej konfiguracji, która pozwala na przetwarzanie danych z dowolnego źródła. System zostanie skonfigurowany, żeby akceptował zestawy danych tylko z serwera na którym jest uruchomiony



Rys. 14 Lista zagrożenia po usunięciu zagrożenia CSP: script-src unsafe-inline

Podobnym zagrożeniem jest „CSP style-src unsafe-inline". Rozwiązanie jest bardzo podobne jak powyżej. Dodano opcję `style-src 'self'` Content-Security-Policy. Flaga ta niweluje zagrożenie, po ponownym uruchomieniu ataku liczba zagrożeń zmniejszyła się do 11 (rysunek 15).

Rys. 15 Lista zagrożeń po usunięciu zagrożenia CSP: style-src unsafe-inline

Ostatni zagrożeniem z listy CSP jest Wildcard Directive. Wpisanie konfiguracji z *Listing 8* do pola Content-Security-Policy wyeliminuje wszystkie zagrożenia z rodziny CSP.

Listing 8 Konfiguracja która niweluje zagrożenie Wildcard Directive

```
img-src 'self'; connect-src 'self'; font-src 'self'; media-src
'self'; manifest-src 'self'; prefetch-src 'self'; form-action 'self'
```

Rysunek 16 prezentuje rezultat skanowania po zastosowaniu powyższych instrukcji. Najważniejsza informacją jest, że wszystkie zagrożenia z rodziny CSP zostały wyeliminowane. Aplikacja jest teraz bardziej bezpieczna.

Rys. 16 Lista zagrożenia po usunięciu wszystkich zagrożeń z rodziny CSP

Listing 9 przedstawia ostateczną konfigurację, którą należy użyć w celu zabezpieczenia systemu przed podatnościami typu CSP. Każdy parametr posiada flagę `self`, która sprawia, że tylko zasoby pochodzące z hosta, czyli serwera na którym uruchomiona jest aplikacja, będą przetwarzane przez oprogramowanie. Rozwiązanie to zabezpiecza przed niechcianymi atakami, ponieważ zainfekowany zasób nie zostanie przetworzony.

Listing 9 Ostateczna konfiguracja, która w pełni niweluje podatności z rodziny CSP

```
frame-src 'self'; frame-ancestors 'self'; object-src 'none'; script-src
'self'; style-src 'self'; img-src 'self'; connect-src 'self'; font-src
'self'; media-src 'self'; manifest-src 'self'; prefetch-src 'self';
form-action 'self'
```

Następne poważne zagrożenie to „Absence of Anti-CSRF Tokens” Podatność ta jest bardzo niebezpieczna, ponieważ może doprowadzić do wywołania niechcianych akcji przez użytkownika, który o tym nie wie. Brak tokenu CSFR powoduje, że system nie jest w stanie odróżnić, które żądanie jest wykonywane przez użytkownika, który chciał wykonać tą akcję. Atakujący nie jest w stanie podejrzeć odpowiedzi po udanym ataku. Natomiast może wykorzystać tą podatność do wysłania z oficjalnego konta organizacji wiadomości email z wirusem, dlatego zagrożenie ta jest bardzo niebezpieczne i należy się go pozbyć jak najszybciej. Wyeliminować można tą podatność przez losowe generowanie tokenu w aplikacji i przesyłanie tego tokenu do użytkownika. Każde żądanie, które zostanie wysłane do serwera zostanie przetworzone tylko w przypadku, jeśli token przesłany przez użytkownika zgadza się z tokenem, który przechowuje serwer. Wprowadzenie prostego tokenu sprawia, że serwer jest w stanie rozróżnić, które żądania pochodzą od użytkownika, a które to potencjalne ataki, których nie należy przetwarzać. Dokumentacja Keycloak zawiera informację, że wszystkie logowania do systemu są w pełni bezpieczne, ponieważ oprogramowanie posiada zaimplementowany mechanizm niwelujący podatność CSRF Token.

OWASP ZAP zgłosił tą podatność, ponieważ nie znalazł jednego z poniższych popularnych nazw tokenów w formularzu

- anticsrf

- CSRFToken
- __RequestVerificationToken
- csrfmiddlewaretoken
- authenticity_token
- OWASP_CSRFTOKEN
- anoncsrf
- csrf_token
- _csrf, _csrfSecret
- __csrf_magic
- CSRF
- _token
- _csrf_token

Ważne jest, żeby każdą zgłoszoną podatność rozpatrzyć indywidualnie. Przypadek powyżej przedstawia sytuacje w której oprogramowanie testujące zgłasza podatność. Dzieje się tak ponieważ narzędzia takie jak OWASP ZAP nie są w stanie oszacować czy zgłoszona podatność jest niebezpieczna bez znajomości kontekstu. Tester powinien każdorazowo oszacować, czy podatność zagraża bezpieczeństwu obecnego rozwiązania. Poniżej na *Listing 10*, prezentacja dwóch z kilkudziesięciu żądań, które są wysyłane do serwera podczas próby logowania.

Listing 10 Teść żądań przesyłanych do Keycloak podczas logowania

```
<form id="kc-form-login" onsubmit="login.disabled = true; return true;"
action="http://localhost:8082/realms/gdma/login-
actions/authenticate?session_code=CtPNk9uYR61FBKZwuVzJ57vNEeHIk6MTuBtO3RFqJTs&amp;execution=
0612ab3f-7a49-4b3a-9922-8e941891b87f&amp;client_id=front-end-
localhost&amp;tab_id=9PIQRLYwQwQ" method="post">


 <form id="kc-form-login" onsubmit="login.disabled = true; return true;"
action="http://localhost:8082/realms/gdma/login-actions/authenticate?session_code=-Tkmkj-
0wZfwDFMtaGZKnmxMH1DTPbpvOur3gL-0AH8&amp;execution=0612ab3f-7a49-4b3a-9922-
8e941891b87f&amp;client_id=front-end-localhost&amp;tab_id=_YIqaf4XPyA" method="post">
```

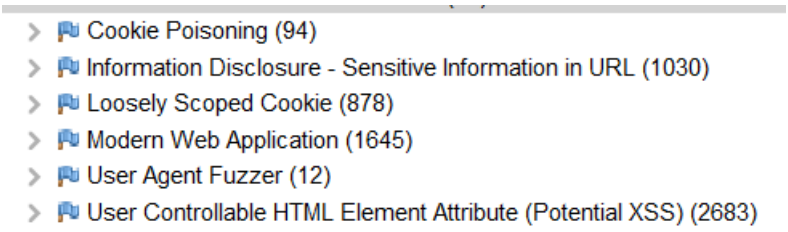
Przy każdym wywołaniu istnieje parametr `session_code`, który przy każdorazowym wygenerowaniu formularza jest inny. Rozwiązanie to w pełni chroni przed podatnością CSRF token, ponieważ atakujący nie jest w stanie zgadnąć tego tokenu i dostarczyć adres url z poprawnym `session_code`.

Kolejnym nieistotnym błędem jest „Cookie with SameSite Attribute None”, ponieważ to cookie nie zawiera żadnych wrażliwych informacji. Używane jest przez Javascript adapter po

stronie web klienta do sprawdzania czy sesja istnieje. Cookie przechowujące wrażliwe dane to „KEYCLOAK_IDENTITY”, które jest odpowiednio zabezpieczone przez flagę SameSite=lax.

Podobnym błędem jest „Cookie without SameSite Attribute”.OWASP ZAP zgłosił ten błąd, ponieważ przy cookie nie ma flagi SameSite. Następujące cookie nie mają tej flagi, są to „KEYCLOAK_LOCALE”, „AUTH_SESSION_ID_LEGACY”. Brak tej flagi nie sprawia, że aplikacja jest podatna na ataki, ponieważ te cookie nie przechowują wrażliwych danych. Pierwsze cookie przechowuje wybrany język w jakim będzie wyświetlany interfejs dla użytkownika. Natomiast drugie jest używane przez web klienta w starszych wersjach adaptera w celu sprawdzenia czy sesja istnieje.

Analizy zagrożeń, które mogą udostępniać informacje na temat aplikacji rysunek 17.



Rys. 17 Zagrożenia, które eksponują informacje o aplikacji

Pierwszą podatnością z listy jest „Cookie Poisoning”, które pozwala nadpisać parametr z żądania url. Robi się to najczęściej poprzez dodanie średnika na końcu zapytania i skopiowaniu parametru z nową wartością. W zależności od użytego serwera, który używa tego parametru zachowanie może być różne. Jedne serwery zignorują dopisany parametr inne z kolei mogą nadpisać istniejącą wartość lub zrobić listę z tego parametru. W przypadku aplikacji keycloak podatność ta nie jest niebezpieczna, ponieważ parametrem jest „kc_locale”, który odpowiada za zwracanie danych w odpowiednim języku. Parametr ten nie przetwarza żadnych wrażliwych danych, przez co aplikacja jest bezpieczna.

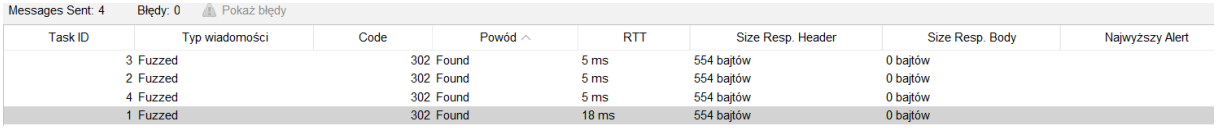
Kolejnym zgłoszonym ostrzeżeniem jest „Information Disclosure – Sensitive Information in URL”. OWASP ZAP informuje, że właściwość session_code zawiera wrażliwe dane na temat sesji, jest to nieprawda. Zgłoszona podatność nie jest niebezpieczna, ponieważ wartość ta jest używana w celu zapobiegnięcia atakom CSRF.

Zagrożenia „Loosely Scoped Cookie” występuje, jeśli nie zdefiniuje się poprawnie zakresu cookie. Zakres można definiować do parent domeny np. parent.com. Cookie stworzone w ten sposób może być wykorzystane przez subdomeny np. www.chil.parent.com. Dobrą praktykom jest definiowanie zakresu, który dokładnie określa domenę i jest jednoznaczny. Podatność ta zostanie wyeliminowana, jeśli aplikacja zostanie opublikowana w sieci i otrzyma adres URL, który jednoznacznie będzie identyfikował Keycloaka. Testując lokalnie podatność tą można wyeliminować poprzez stworzenie aliasu w pliku ect np. 127.0.0.1 keycloak.local. Dzięki temu rozwiązaniu ZAP nie zwraca ostrzeżenia ponieważ teraz witryna nazywa się keycloak.local a nie jak wcześniej localhost.

Najmniej istotnym zagrożeniem jest „Modern Web Application”. OWASP ZAP informuje, że atakowana aplikacja jest stosunkowo nowa i używa nowoczesnych rozwiązań. Nie ma możliwości wyeliminowania tej podatności.

Ostatnim potencjalnym zagrożeniem jest „User Agent Fuzzer”. Nagłówek „User-Agent” jest wysyłany przez przeglądarkę do serwera i zawiera informacje o systemie operacyjnym oraz o przeglądarce, z której korzysta użytkownik. Narzędzie OWASP ZAP wysyła kilkanaście żądań z rożnymi nagłówkami używanej przeglądarki w celu sprawdzenia czy serwer zwróci inna odwiedź, przez co taka informacja może zostać wykorzystana podczas ataku. Serwer Keycloak zawsze zwraca odpowiedź w tym samym kształcie przez co atakujący nie może użyć nagłówka „User-Agent” do ataku na aplikacje.

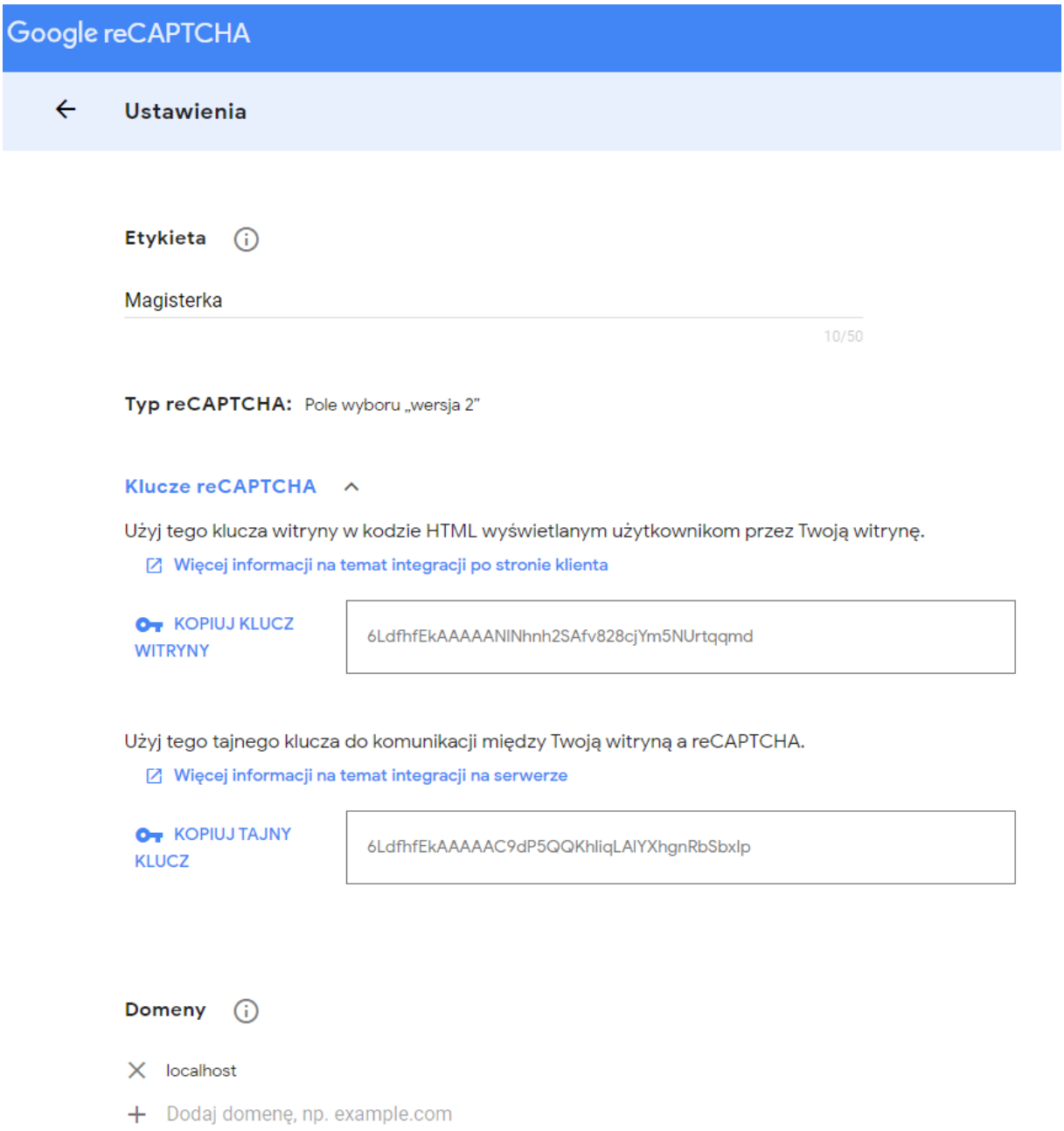
Problem związany z kontami typu bot jest obecnie bardzo powszechny. Keycloak umożliwia włączenie weryfikacji email, które polega na wysłaniu użytkownikowi link na podany email. Link ten po kliknięciu potwierdza, że email istnieje i odblokowuje konto. Kolejnym testem będzie próba stworzenia kilkunastu kont, przy pomocy narzędzia OWASP ZAP. Podobnie jak wcześniej przygotowano atak Fuzzer i go wykonano. Rezultat jest dokładnie taki sam jak przy próbie łamania hasła. Keycloak wykrył nową sesję i zwraca nową stronę z formularze do logowania. Rozwiązanie to zabezpiecza przed tworzeniem dużej ilości kont.


Messages Sent: 4 Błędy: 0 Pokaż błędy

Task ID	Typ wiadomości	Code	Powód	RTT	Size Resp. Header	Size Resp. Body	Najwyższy Alert
3	Fuzzed	302	Found	5 ms	554 bajtów	0 bajtów	
2	Fuzzed	302	Found	5 ms	554 bajtów	0 bajtów	
4	Fuzzed	302	Found	5 ms	554 bajtów	0 bajtów	
1	Fuzzed	302	Found	18 ms	554 bajtów	0 bajtów	

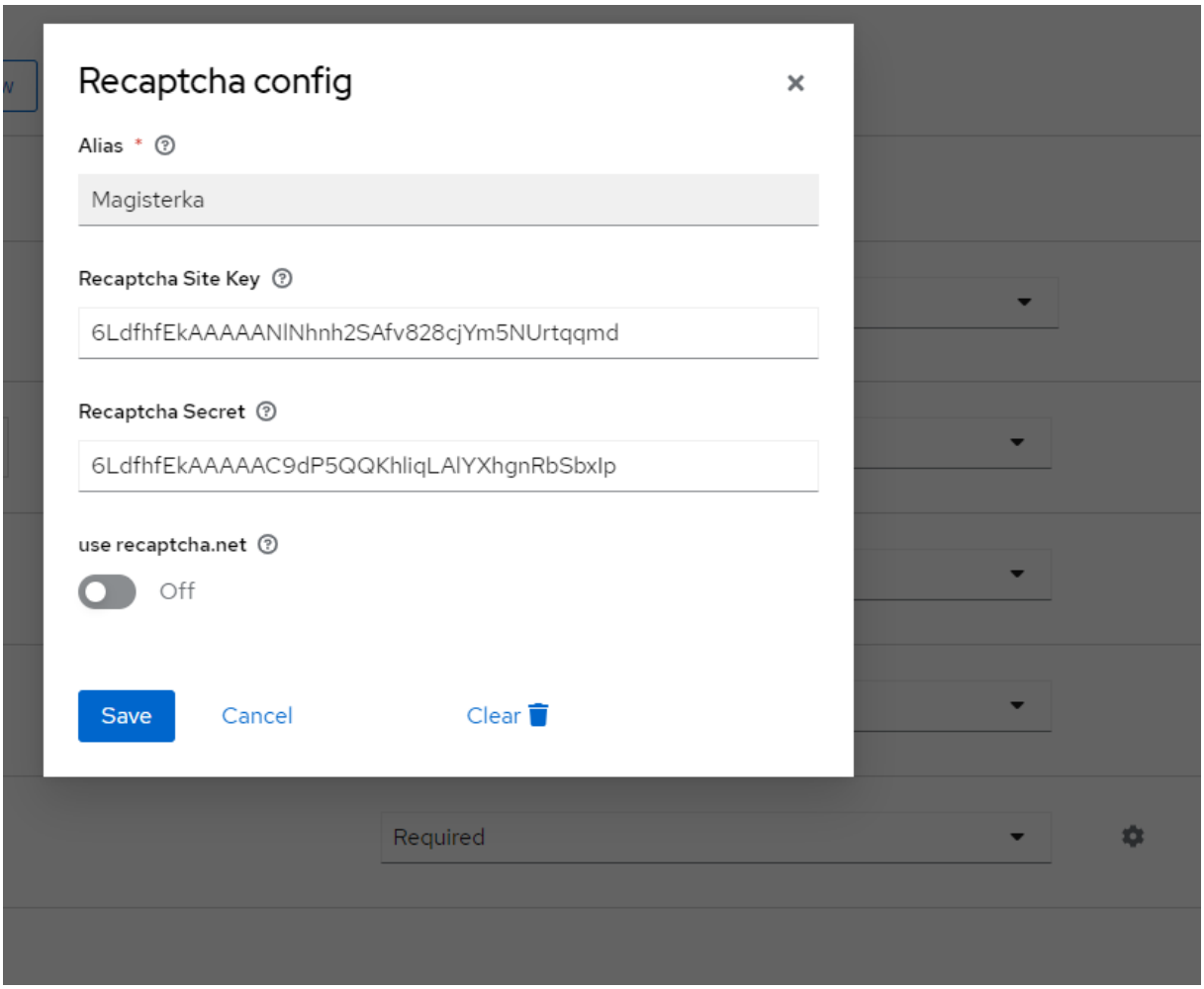
Rys. 18 Wynik po próbie utworzenia 4 użytkowników

Keycloak posiada również możliwość skonfigurowania mechanizmu Recaptcha dla formularza rejestracji użytkowników. W celu skonfigurowania tej funkcjonalności konieczne jest utworzenia konta w Google. Minusem jest, że nie jest wspierana najnowsza implementacja reCAPTCHA tylko wersja 2 z roku 2014. Na GitHubie jest stworzone zadanie na wspieranie najnowszego mechanizmu reCAPTCHA.



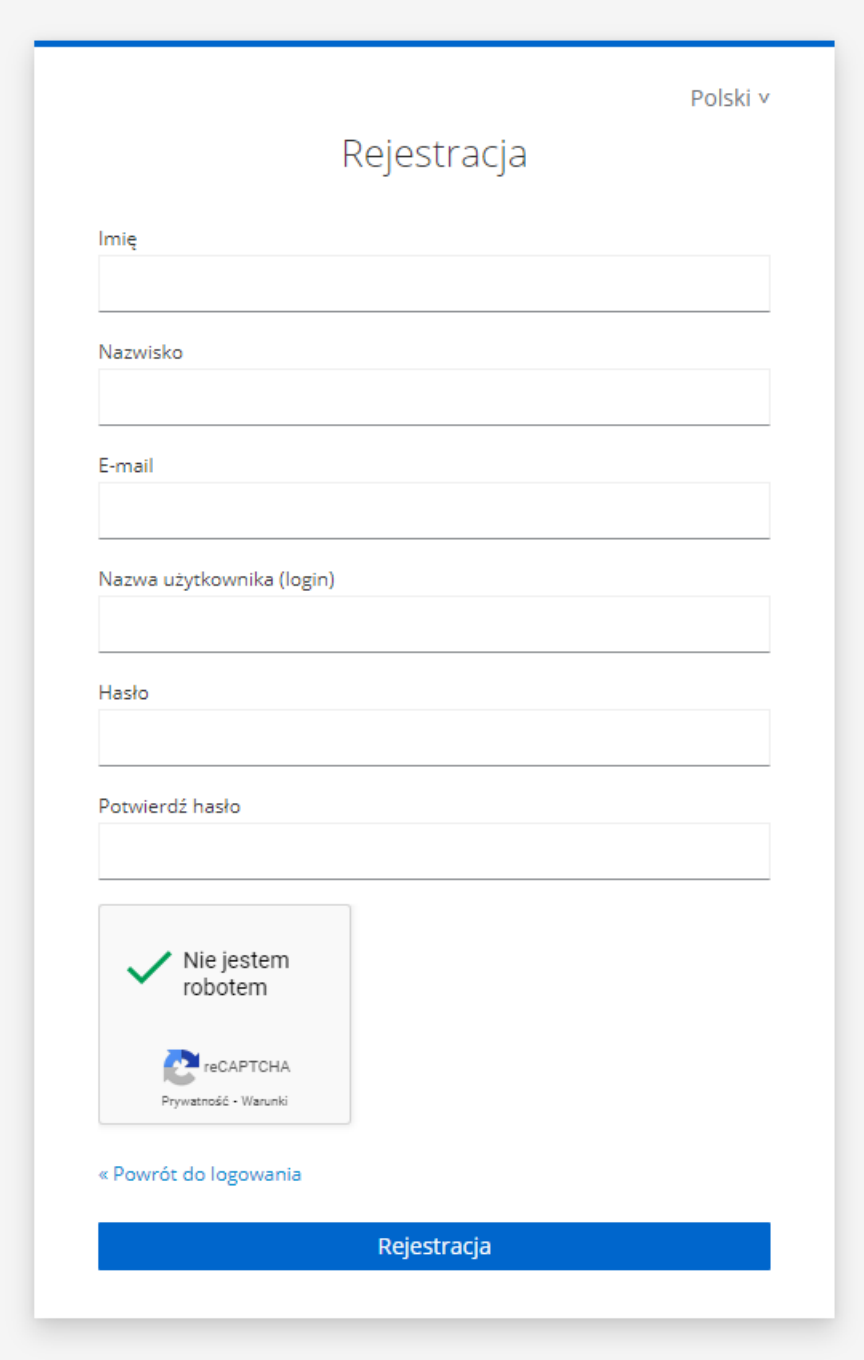
Rys. 19 Wygenerowanie poświadczenie dla aplikacji Keycloak

Kolejno skonfigurowano mechanizm recaptcha w Keycloak. Po zalogowaniu się na konto administratora i aktywacji mechanizmu. Skopiowano klucz witryny oraz prywatny klucz, który będzie służył do rozkodowania przesłanych informacji. Kolejno w usłudze Google dodano te klucze oraz ustawiono domenę na localhost, ponieważ aplikacja była testowana na lokalnym komputerze.



Rys. 20 Skonfigurowany i włączony mechanizm Recaptcha

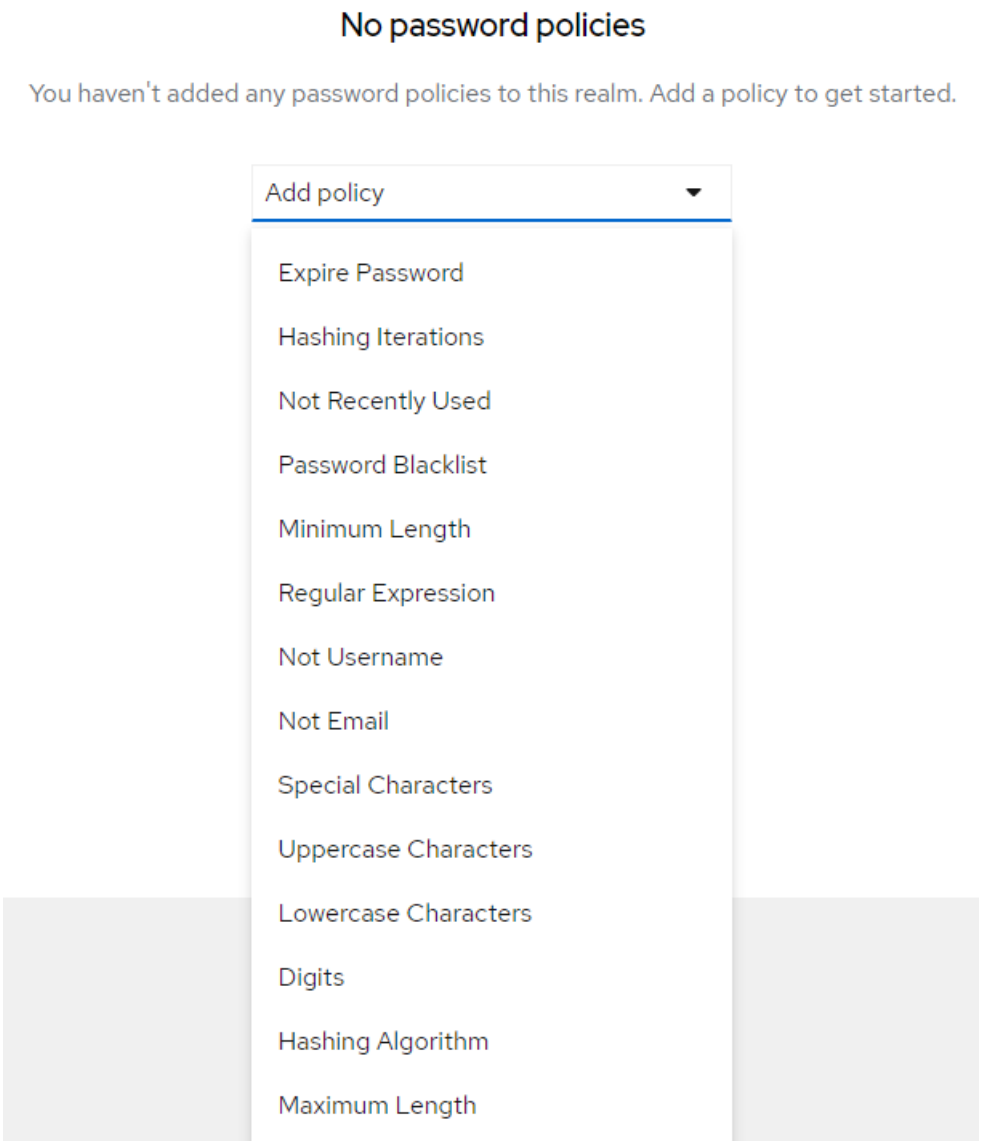
Rejestrując nowego użytkownika, system będzie wymagał zaznaczenia opcji „Nie jestem robotem”. Proste skrypty zakładające fikcyjne konta nie będą już w stanie tego robić, ponieważ trzeba wybrać opcję, że nie jest się robotem. Poniżej zaprezentowano widok najnowszego formularza rejestracji, który wymaga dodatkowego potwierdzenia przez użytkownika.



Rys. 21 Widok formularza rejestracji

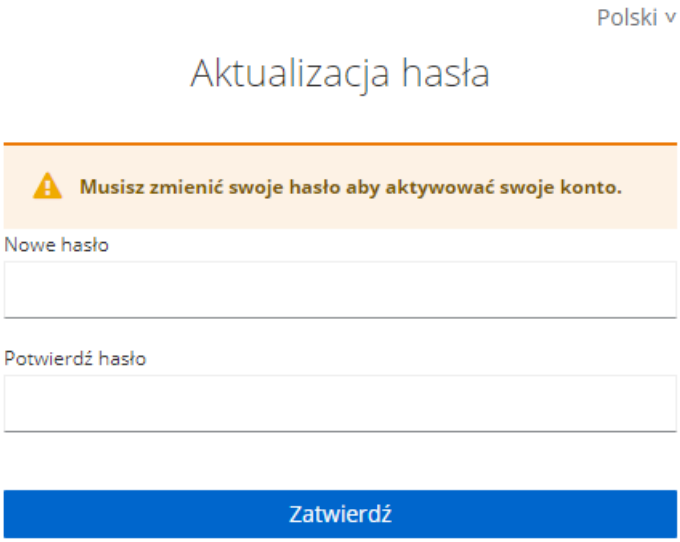
3.4 Polityki Haseł w Keycloak

Keycloak posiada również możliwość skonfigurowania polityki haseł (rysunek 22). Domyślne ustawienia nie mają żadnej polityki, jest to niebezpieczne, ponieważ użytkownik może utworzyć hasło, które ma tylko jeden znak. Tabela 1 przedstawia, ile potrzeba czasu, aby złamać hasło. Domyślna konfiguracja nie jest bezpieczna, dlatego warto określić długość, ilość małych i wielkich liter oraz liczbę znaków specjalnych. Pozwoli to zabezpieczyć przyszłych użytkowników aplikacji przed atakami typu brute force.



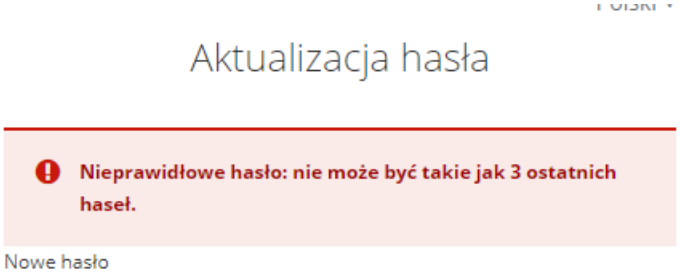
Rys. 22 Zbiór polityk, które można skonfigurować

Pierwszym elementem na liście jest „Expire Password”, opcja ta ustawia przez jaki czasu użytkownik może posługiwać się swoim hasłem, jeżeli okres ten minie zostanie zmuszony do aktualizacji hasła przy najbliższym logowaniu. Domyślna wartość wynosi 365 dni. Wartość została ustawiona na 0 podczas najbliższego logowania, po podaniu odpowiedniego loginu i hasła użytkownik zostanie poproszony o zmianę hasła (rysunek 23).



Rys. 23 Aktualizacja nieaktywnego hasła

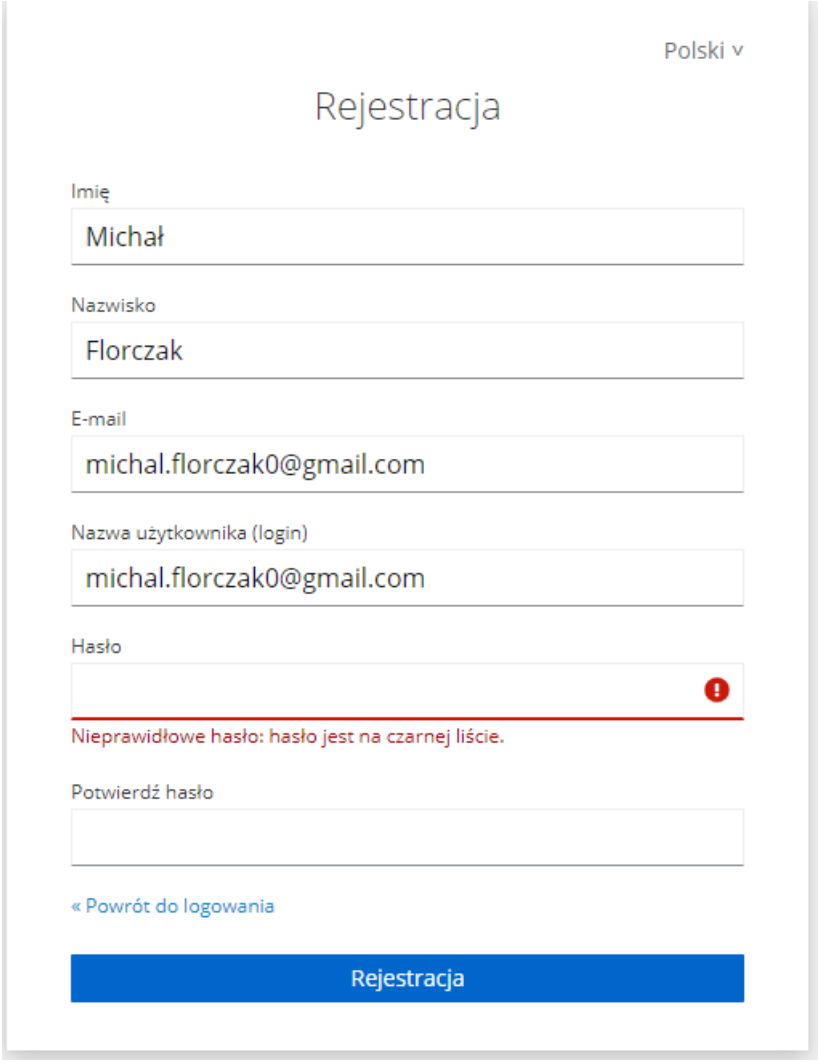
„Not Recently Used" Keycloak posiada mechanizm, który przechowuje również historyczne hasła. Domyślnie to ustawienie blokuje próbę zmiany hasła, jeżeli hasło jest takie same jak 3 ostatnie hasła (rysunek 23). Wartość tą można zmienić.



Rys. 24 Próba zmiany na poprzednie hasło

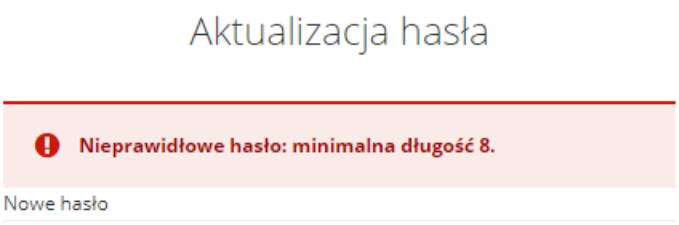
„Hashing Iterations" jest to liczba, ile razy hasło zostanie za haszowane przed weryfikacją lub zapisaniem do bazy. Domyślna wartość wynosi 27500. Większa wartość powodować będzie, że hasło będzie trudniejsze do złamania. Minusem jest wolniejsze działanie aplikacji, ponieważ więcej iteracji trzeba wykonać do odszyfrowania hasła.

„Password Blacklist" funkcjonalność ta pozwala wskazać plik tekstowy, w którym znajdują się najbardziej popularne hasła (rysunek 25). Keycloak nie pozwoli stworzyć konta lub zmienić hasła na hasło z listy. Dokumentacja wprowadza w błąd, ponieważ umieszczenie pliku z hasłami w lokalizacji, na którą wskazuje dokumentacja nie działa. Poprawna lokalizacja to „/opt/keycloak/data/password-blacklists/". Informacja ta została znaleziona w jednym z commitów do repozytorium Keycloak.



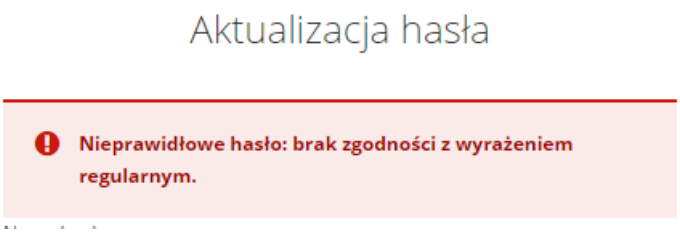
Rys. 25 Komunikat o haśle, które jest na czarnej liście

„Minimum Length” jest to minimalna liczba znaków dla hasła. Domyślnie jest to liczba 8 (rysunek 26).



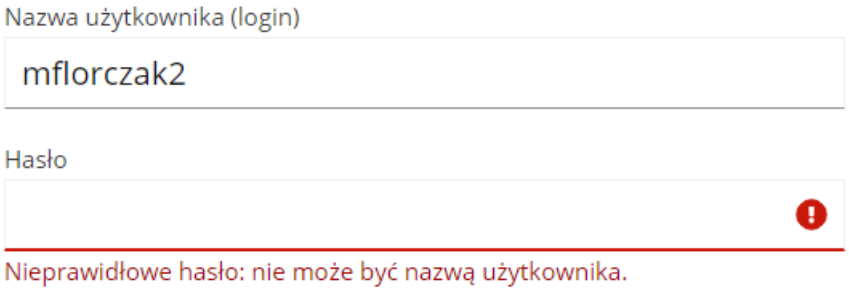
Rys. 26 Próba zmiany hasła na krótsze niż 8

„Regular Expression” opcja ta daje bardzo dużo możliwości, ponieważ można stworzyć dowolną regułę lub kilka reguł, które sprawdzą czy podane hasło jest dokładnie takie jakie chcemy. Dodano wyrażenie regularne „`^[0-9]*$`”, które akceptuje tylko liczby w haśle (rysunek 27).



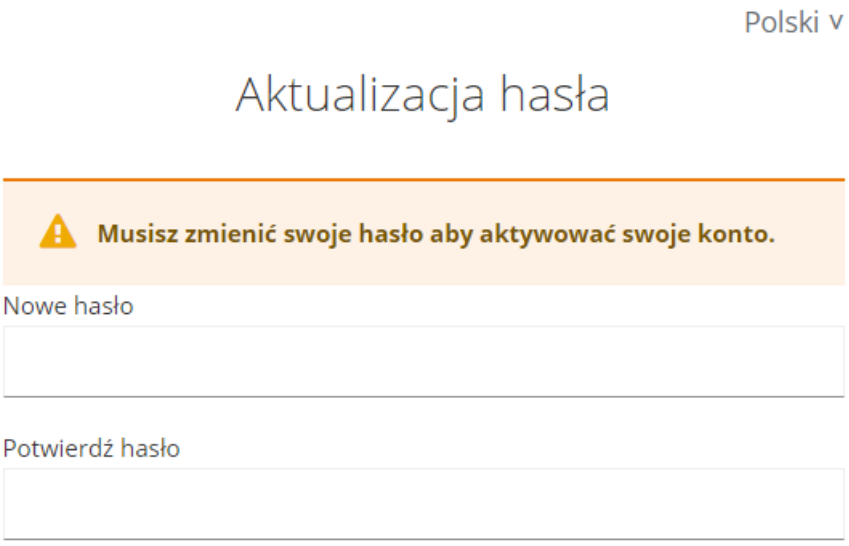
Rys. 27 Brak zgodności z wyrażeniem regularnym

„Not Email" i „Not Username" są to polityki, które wymuszają na użytkownikowi, że hasło musi być różne od adresu mail oraz nazwy użytkownika. Polityki te nie działają w sposób konsystentny. Polityka „Not Username" blokuje możliwość stworzenia konta, jeżeli użytkownik w pole hasła wpisze nazwę użytkownika (rysunek 28).



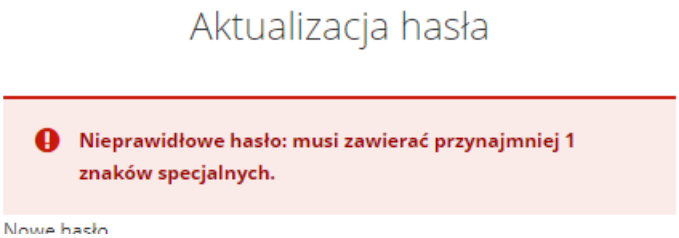
Rys. 28 Hasło takie same jak nazwa użytkownika

Natomiast polityka „Not Email" pozwala utworzyć konto, jeżeli hasło jest takie samo jak adres email. Następnie system zmusza użytkownika do zmiany tego hasła na inne i jeżeli wtedy zostanie podane takie samo hasło (rysunek 29). System zgłosi komunikat o błędnym haśle.



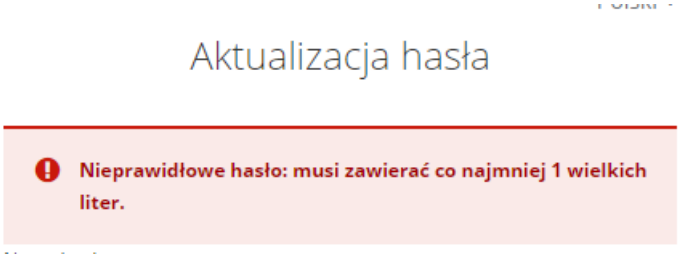
Rys. 29 Komunikat w przypadku, jeżeli hasło jest takie same jak adres email

„Special Characters" opcja ta wymusza, że hasło podane przez użytkownika, musi mieć domyślnie co najmniej jeden znak specjalny (rysunek 30). Wartość tą można zwiększyć.

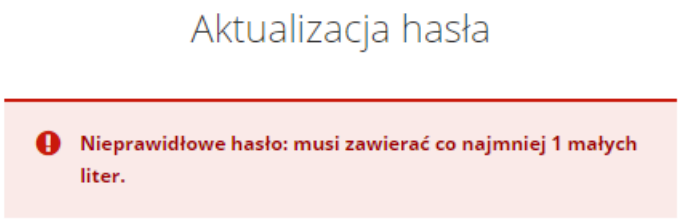


Rys. 30 Komunikat o błędnej ilości znaków specjalnych w haśle

„Uppercase Characters" (rysunek 31) i „Lowercase Characters" (rysunek 32) polityki te określają kolejno liczę wielkich liter w haśle i liczbę małych liter. Domyślnie wartość dla obu wynosi jeden.

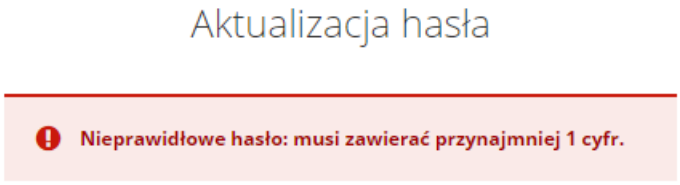


Rys. 31 Komunikat o błędnej ilości wielkich liter w haśle



Rys. 32 Komunikat o błędnej ilości małych liter w haśle

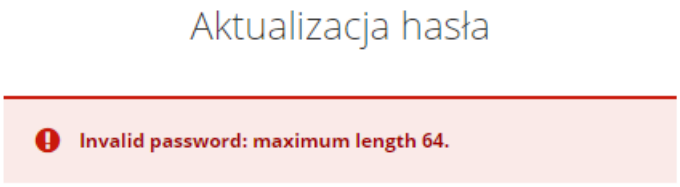
„Digits" opcja ta określa liczbę znaków numerycznych, które musi zawierać hasło (rysunek 33). Domyślnie wymagana liczba cyfr po aktywacji wynosi jeden.



Rys. 33 : Komunikat o błędnej ilości cyfr w haśle

„Hash Algorithm" Keycloak implementuje tylko jeden algorytm haszujący dla haseł, jest to PBKDF2. Istnieje możliwość dodania własnego algorytmu haszującego, ale wiąże się to z własną implementacją i użyciem SPI (Service Provider Interface). Keycloak umożliwia nadpisanie „Service Provider" i dzięki temu dodanie własnej logiki biznesowej.

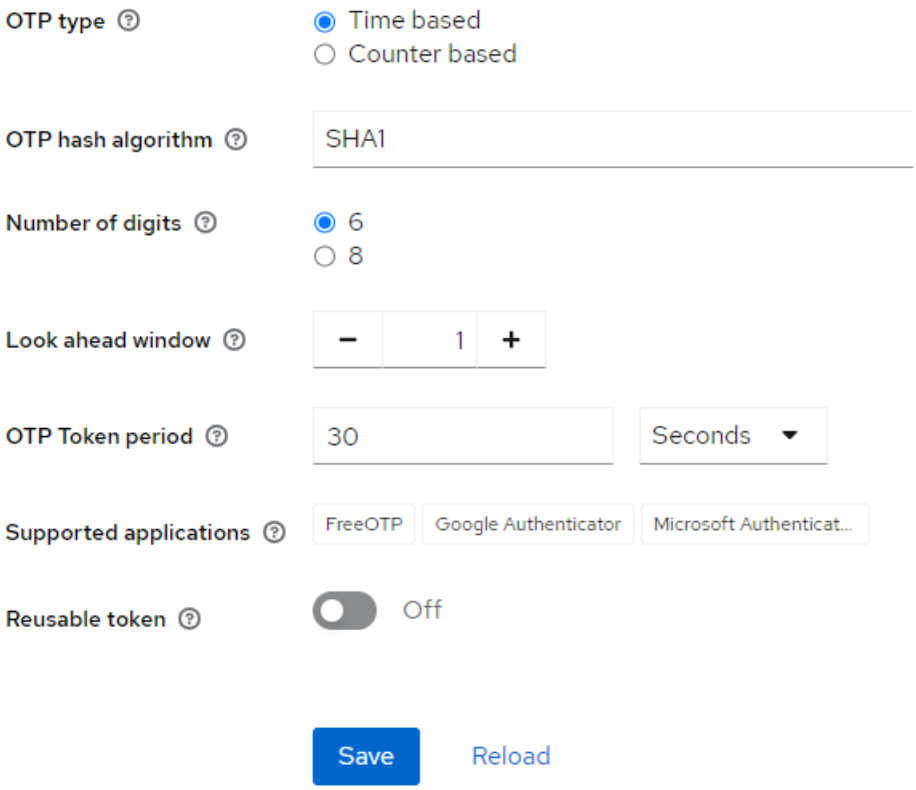
„Maximum Length" polityka ta określa maksymalną długość hasła domyślnie opcja ta po wybraniu pozwoli na utworzenie hasła, którego długość nie może przekraczać 64 znaków. Podczas testowania okazało się, że funkcjonalność działa, ale nie została dostarczona internacjonalizacja dla języka polskiego. Komunikat jest w języku angielskim (rysunek 34).



Rys. 34 Komunikat informujący, że hasło jest za długie

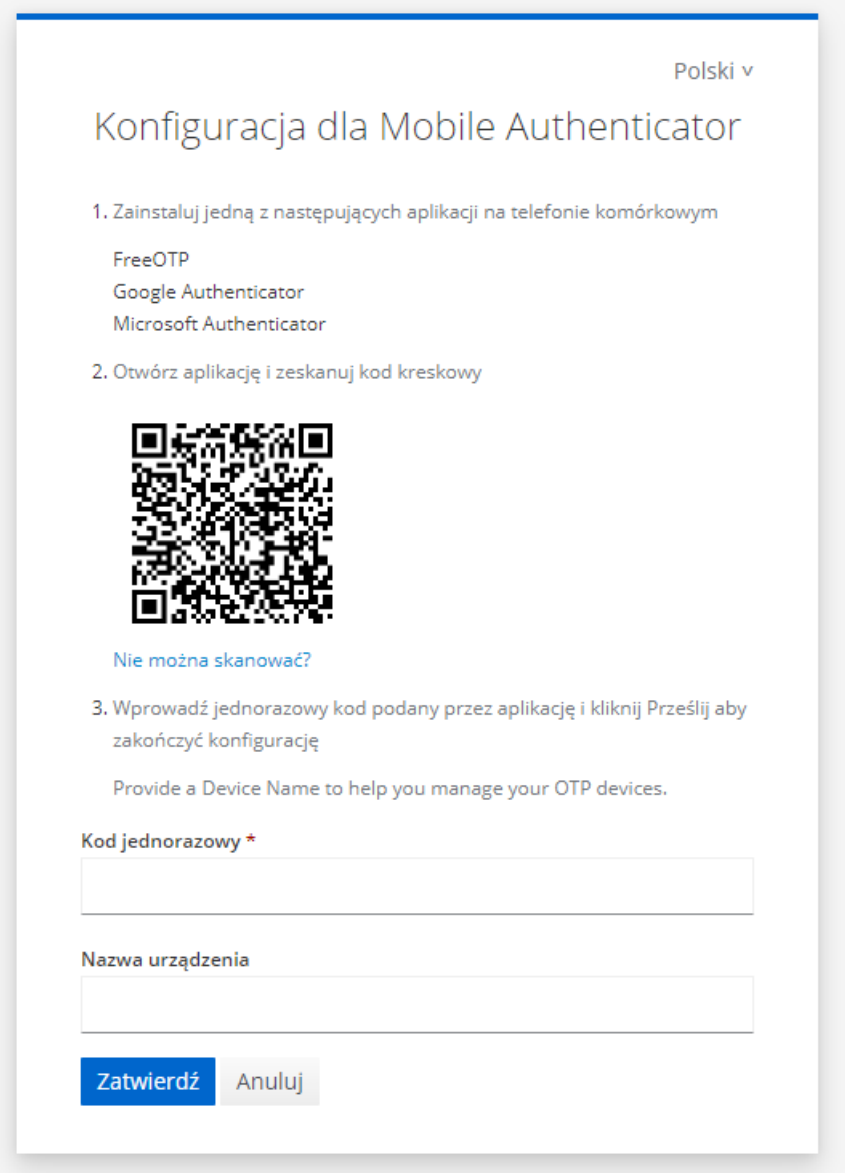
Keycloak posiada również bardzo ważny mechanizm OTP, który może zostać wykorzystany przez użytkownika systemu w celu zwiększenia bezpieczeństwa (rysunek 35). Skrót ten

oznacza „One Time Password”, czyli tłumacząc na język polski, hasło jednorazowe. Rozwiązanie to zwiększa drastycznie bezpieczeństwo, ponieważ nawet jeżeli hasło zostanie złamane, to system poprosi o wpisanie jednorazowego kodu wygenerowanego przez aplikacje do uwierzytelniania. Keycloak wspiera trzy mechanizmy, są to kolejno FreeOTP od RedHut, Google Authenticator i Microsoft Authenticator. Aktywując mechanizm OTP w 100% wyeliminowana zostanie podatność na ataki typu Brute Force.



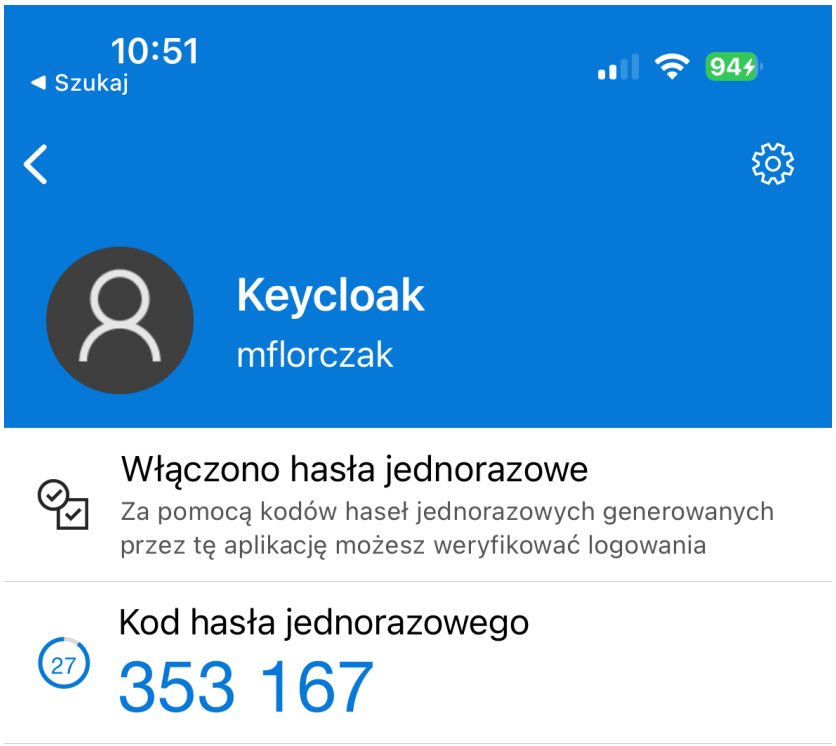
Rys. 35 Widok możliwych opcji do skonfigurowania w mechanizmie OTP

Domyślnym algorytmem haszujący jest SHA1 dla OTP. Istnieje możliwość zmiany na SHA256 i SHA512. Mechanizm OTP generuje cyfry jako hasło jednorazowe. Administrator może określić długość kodu. Domyślnie jest to 6 cyfr. Istnieje możliwość wybrania 8 cyfr, ale korzystanie z systemu będzie bardziej uciążliwe, ponieważ każdorazowo użytkownik, który próbuje się uwierzytelnić, zostanie zmuszony do wpisania 8 cyfrowego kodu. Aktywacja tej opcji przez użytkownika jest bardzo prosta. Wystarczy na urządzenie mobilne pobrać jedną z wymienionych wcześniej aplikacji i zeskanować kod QR (rysunek 36).



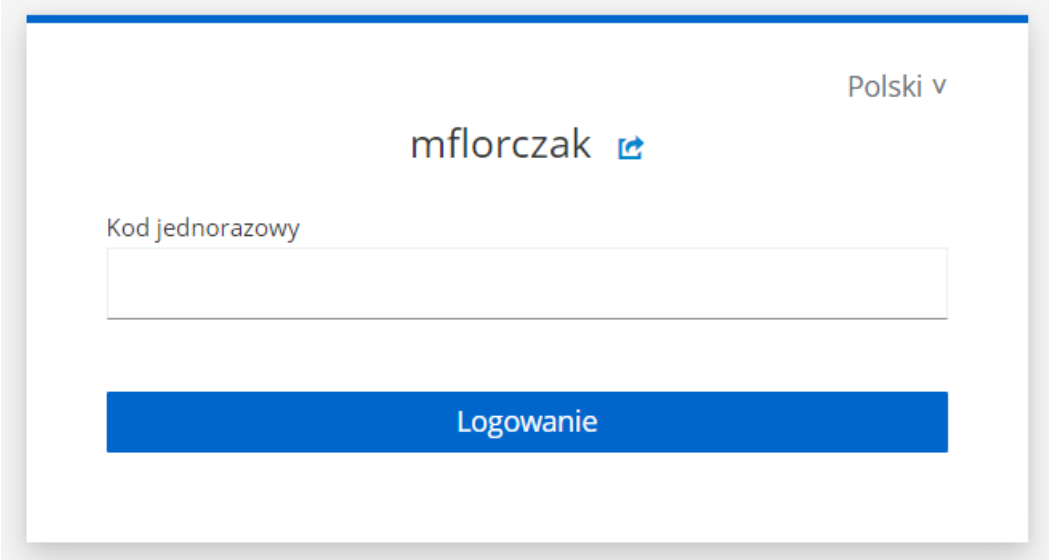
Rys. 36 Widok konfiguracji generatora haseł OTP

Następnie w aplikacji wygeneruje się hasło jednorazowe, trzeba je wpisać w pole Kod jednorazowy. Nazwa urządzenia jest opcjonalna, ale warto podać, jeżeli planuje się używać więcej niż jednego mechanizmu OTP. Pozwoli to w przyszłości na łatwiejszą identyfikację, aplikacji której należy użyć.



Rys. 37 Widok z aplikacji Microsoft Authenticator

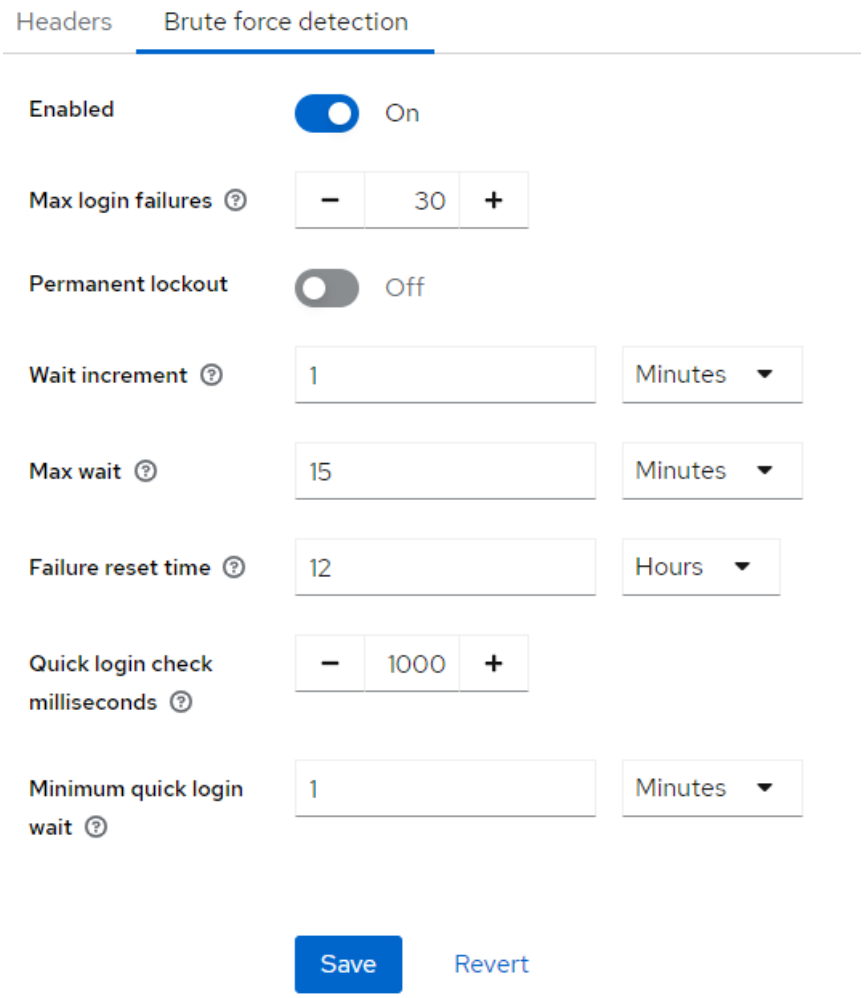
Zalogować się do konta można tylko po podaniu dwóch poprawnych haseł. Pierwsze hasło to hasło do konta użytkownika, po podaniu poprawnego hasła Keycloak przekieruje użytkownika na nowy widok, gdzie zostanie poproszony o wpisanie hasła jednorazowego z aplikacji uwierzytelniającej połączonej z jego kontem (rysunek 37). Wchodząc do aplikacji mobilnej, będzie można uzyskać hasło jednorazowe. Należy je wpisać w pole Kod jednorazowy (rysunek 38). W przypadku jeżeli hasło będzie takie same, system zezwoli na zalogowanie się na konto użytkownika.



Rys. 38 Widok okna mechanizmu walidacji OTP

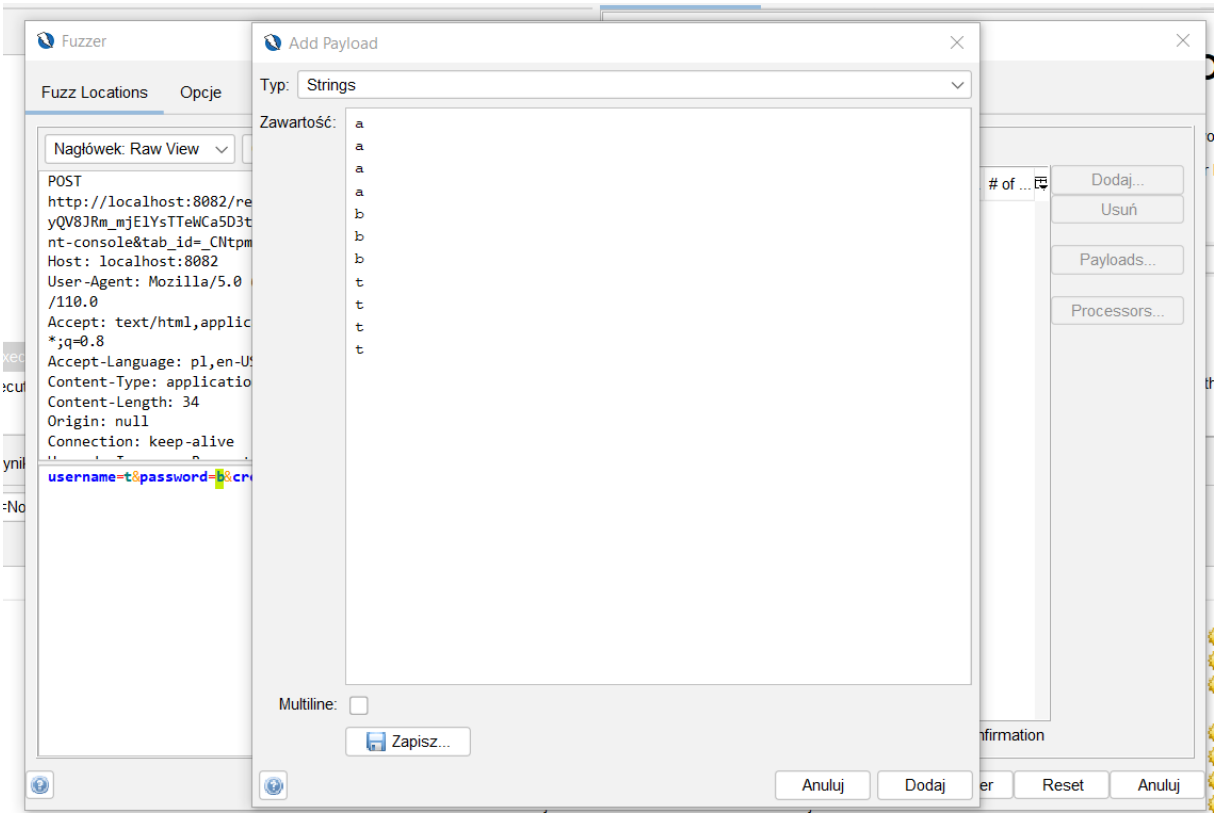
3.5 Brute force

Keycloak posiada również mechanizm, który umożliwia wykrycie ataków Brute force (rysunek 39). Istnieje możliwość konfiguracji po jakiej ilości niepoprawnych logowań zablokować konto. Administrator może również określić, czy konto ma być blokowane czasowo czy na stałe.



Rys. 39 Domyśla konfiguracja wykrywania Brute force

Ostatnie dwie opcje tyczą się ataków Brute force przy pomocy zewnętrznego oprogramowania. Opcja „Quick login check milliseconds" pozwala skonfigurować odstępy czasowe pomiędzy kolejnymi próbami logowania. System zablokuje konto użytkownika na jedną minutę, jeżeli wykryje próbę ataku. Poniżej test tej funkcjonalności przy pomocy narzędzia OWASP ZAP. Przechwycono żądanie POST, które odpowiedzialne jest za uwierzytelnienie użytkownika. Uruchomiono atak typu Fuzzer z narzędzia OWASP ZAP .Rysunek 40przedstawia konfigurację słownika dla haseł.



Rys. 40 Konfiguracja słownika do łamania hasła

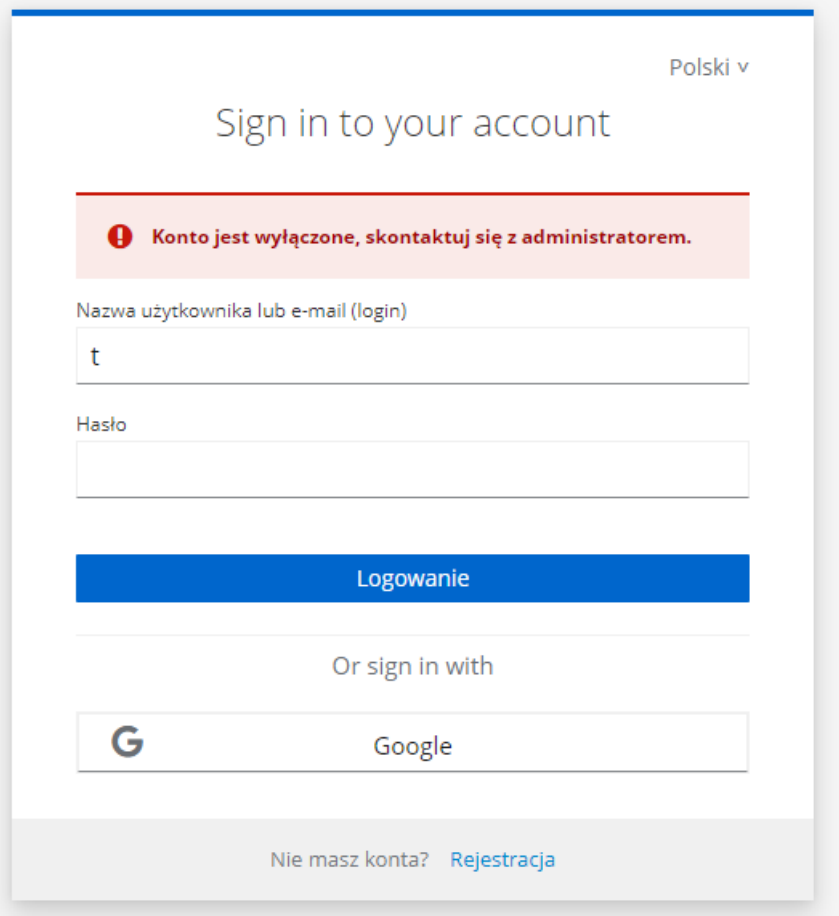
Na poniższym rysunku przedstawiono wynika powyższego ataku. Keycloak nie pozwolił wykonać tego ataku, nie wykonała się ani jedna próba ataku. Przy każdej próbie jest zwracana odpowiedź z kodem 302 i adresem url do strony logowania. Przy pomocy narzędzia OWASP ZAP nie jesteśmy w stanie dokonać ataku typu Brute Force na aplikacje. Wielkość odpowiedzi jest zawsze taka sama, nawet jeśli atakujący podał poprawne hasło, system Keycloak zawsze zwraca stronę do logowania w odpowiedzi, ponieważ wykrył próbę autoryzacji z innego miejsca niż wcześniej.

Task ID	Typ wiadomości	Code	Powód	RTT	Size Resp. Header
1	Fuzzed	302	Found	8 ms	554 bajtów
3	Fuzzed	302	Found	10 ms	554 bajtów
9	Fuzzed	302	Found	9 ms	554 bajtów
2	Fuzzed	302	Found	10 ms	554 bajtów
10	Fuzzed	302	Found	9 ms	554 bajtów
6	Fuzzed	302	Found	9 ms	554 bajtów
7	Fuzzed	302	Found	9 ms	554 bajtów
8	Fuzzed	302	Found	9 ms	554 bajtów
4	Fuzzed	302	Found	10 ms	554 bajtów
5	Fuzzed	302	Found	15 ms	554 bajtów
11	Fuzzed	302	Found	16 ms	554 bajtów
12	Fuzzed	302	Found	15 ms	554 bajtów
13	Fuzzed	302	Found	11 ms	554 bajtów

Rys. 41 Wynik ataku Brute force

Keycloak nie informuje użytkownika, jeżeli jego konto zostało zablokowane tymczasowo jest to poprawne zachowanie, ponieważ w przypadku zgadnięcia hasła, atakujący dostanie

informacje z systemu, że konto zostało zablokowane. Atakujący może poczekać i w przyszłości się zalogować. Oprogramowanie zwraca komunikat, że login lub hasło są nie poprawne. Powodować może to frustrację u użytkownika, ponieważ on wprowadza poprawne hasło i mimo to dostaje komunikat, że hasło jest niepoprawne, gdyby osoba ta dostała informacje, że jego konto zostało zablokowane system dla niego byłby bardziej przejrzysty. Istnieje w oprogramowaniu korelacja, że jeśli system jest bardzo bezpieczny to wymaga na użytkowniku uciążliwych operacji. Inna sytuacja jest natomiast w przypadku, jeśli Administrator włączy opcję permanentnego blokowania konta zamiast tymczasowej blokady. Użytkownik po wpisaniu poprawnego loginu i hasła otrzyma informację, że jego konto zostało zablokowane (rysunek 42).

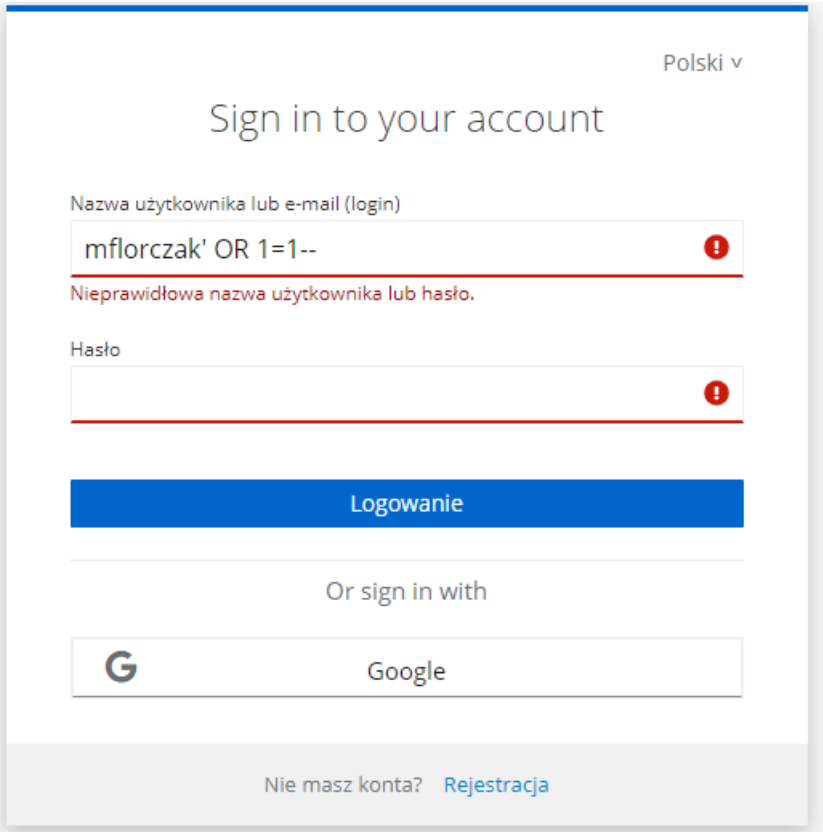


Rys. 42 Komunikat po próbie logowania na konto, które zostało zablokowane na stałe

3.6 SQL Injection

SQL injection jest to niebezpieczna podatność, która umożliwia przy pomocy spreparowanego zapytania SQL wyświetlić dane, do których atakujący nie ma dostępu lub ominąć mechanizm walidacji, który zabezpiecza wrażliwe dane przed nieupoważnionym dostępem. Dokumentacja Keycloka informuje, że obecnie nie ma żadnych znanych podatności SQL injection. Najpierw podjęto próbę wstrzyknięcia SQL przez formularz do logowania, jeżeli uda się zalogować na podanego użytkownika bez znajomości hasła oznacza to, że aplikacja jest podatna na ataki typu SQL injection. Podano nazwę użytkownika i

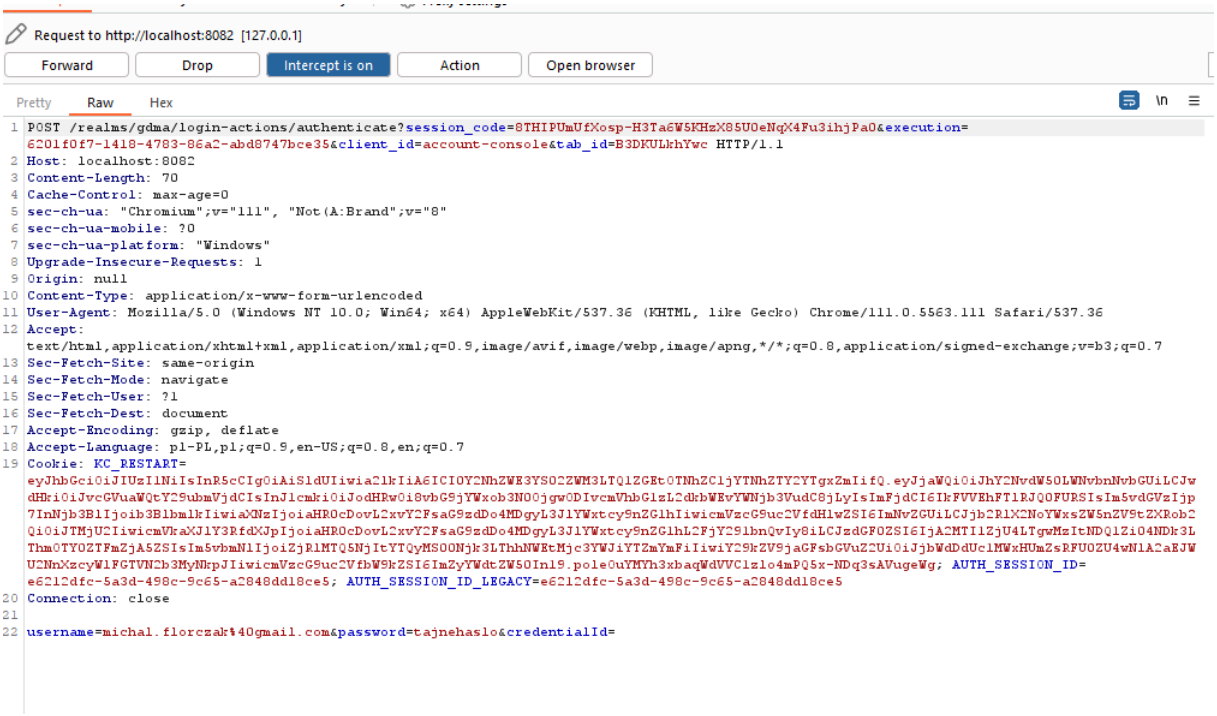
doklejono warunek, który zawsze jest poprawny. Tak spreparowana nazwa użytkownika została przesłana do serwera, jeżeli aplikacja byłaby podatna, to udałoby się zalogować na użytkownika mflorczak. Odpowiedź z serwera natomiast wskazuje, że nie istnieje taki login lub hasło do niego jest nieprawdziwe (rysunek 43). Informuje nas to o tym, że aplikacja nie jest podatna na ataki SQL.



Rys. 43 Rezultat ręcznego ataku SQL injection

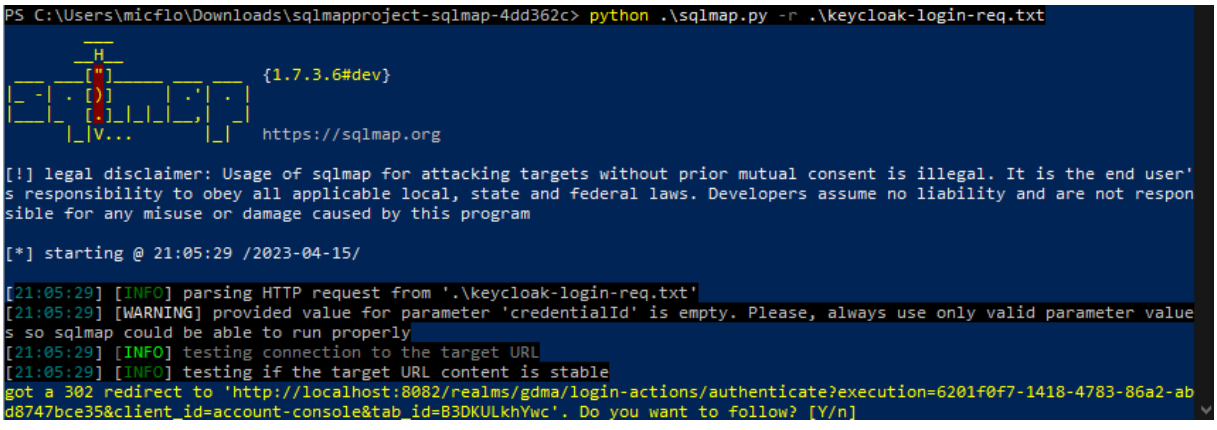
Pisanie takich zapytań ręcznie jest bardzo nieefektywne, ponieważ musimy dostosować zapytanie do konkretnej bazy danych. Istnieje narzędzie sqlmap, które posiada zestaw testów penetracyjnych. Testy te dogłębnie testują aplikacje pod względem tejże podatności.

Nie da się w prosty sposób użyć narzędzia sqlmap na formularzu do logowania, ponieważ w żądaniu przesyłane są ciasteczka oraz nagłówki. Najlepszym sposobem na przechwycenie komunikacji jest użycie narzędzia Burpsuite. Oprogramowanie to pozwala przechwycić, podejrzeć i kolejno zapisać treść do podanego pliku (rysunek 44).



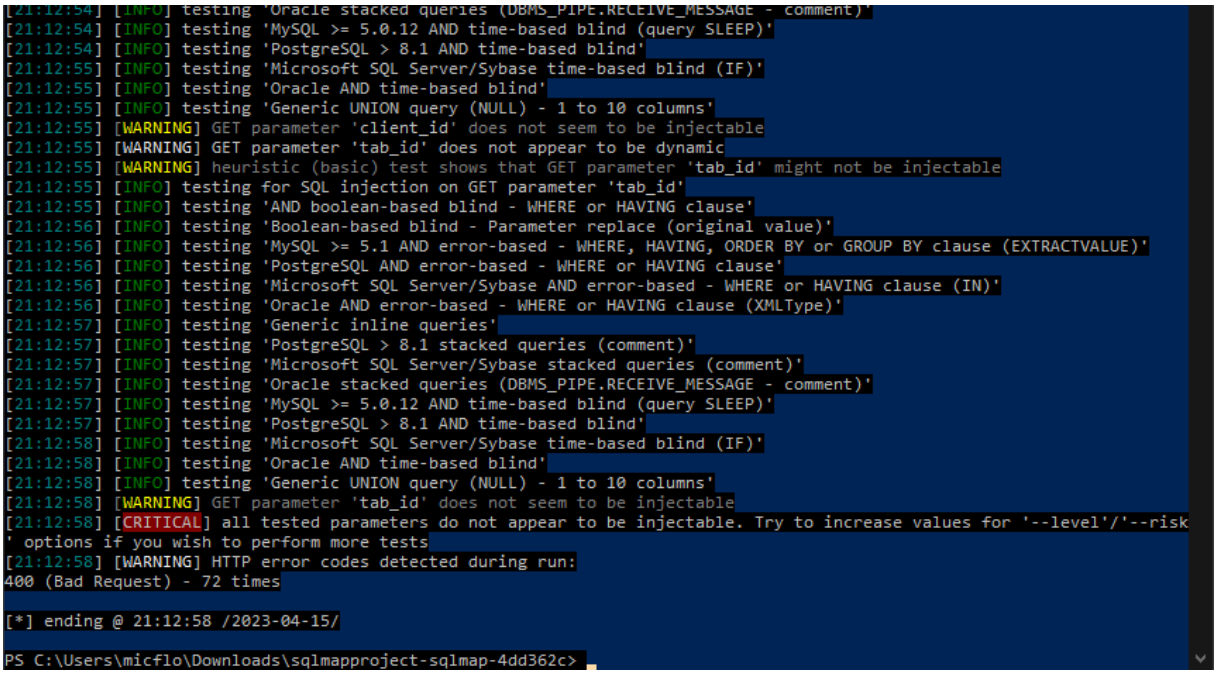
Rys. 44 Treść przechwyconego żądania

Treść tą zapisano do pliku i kolejno jej użyto wywołując sqlmap (rysunek 45). Program informuje, że żądanie nie jest aktualne i aplikacja zwraca nowy adres URL. Zezwolono na przekierowanie na ten nowy adres i rozpoczęto atak.



Rys. 45 Wywołanie testów sqlmap

SQLmap wywołuje wiele testów penetracyjnych, w których najpierw próbuje ustalić jakiej bazy danych używa atakowana aplikacja. Kolejnym krokiem jest przygotowanie dedykowanych ataków do wykrytej wersji bazy danych. Następnie sprawdzane jest, czy w parametry można wstrzyknąć zapytanie SQL (rysunek 46).



Rys. 46 Rezultat nieudanego ataku SQL injection na aplikacje

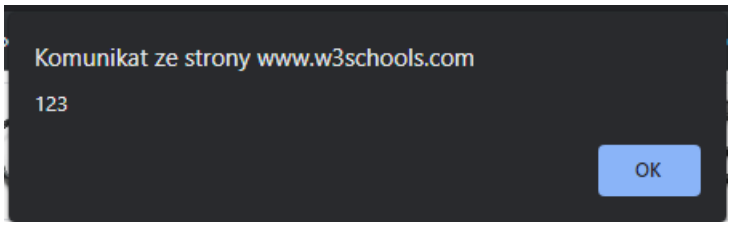
Aplikacja Keycloak nie jest podatna na znane ataki typu SQL injection. Narzędzie sqlmap nie było w stanie określić dostawcy oraz wersji bazy danych. Również nie udało się wykryć parametrów, do których można by wstrzyknąć SQL. Nazwa użytkownika oraz hasło są w pełni chronione przed wstrzykiwaniem zapytań SQL.

3.7 XSS

XSS(Cross-Site-Scripting) jest to podatność, która umożliwia wstrzyknąć JavaScript, który zostanie wykonany w przeglądarce użytkownika który wyświetli zainfekowaną stronę. Podatność ta jest bardzo niebezpieczna, ponieważ atakowana osoba nie musi wchodzić w żaden link wystarczy, że wyświetli zainfekowaną stronę internetową. Atakujący jest w stanie ukraść dane niezbędne do autoryzacji i zalogować się na konto atakowanego użytkownika. Najprostszym sposobem przetestowania, czy strona jest podatna na XSS jest wklejenie w dowolny formularz kodu z *Listing 11* i wysłanie tego formularza. Strona jest podatna na ataki XSS, jeżeli pojawi się alert jak na załączonym rysunku 47.

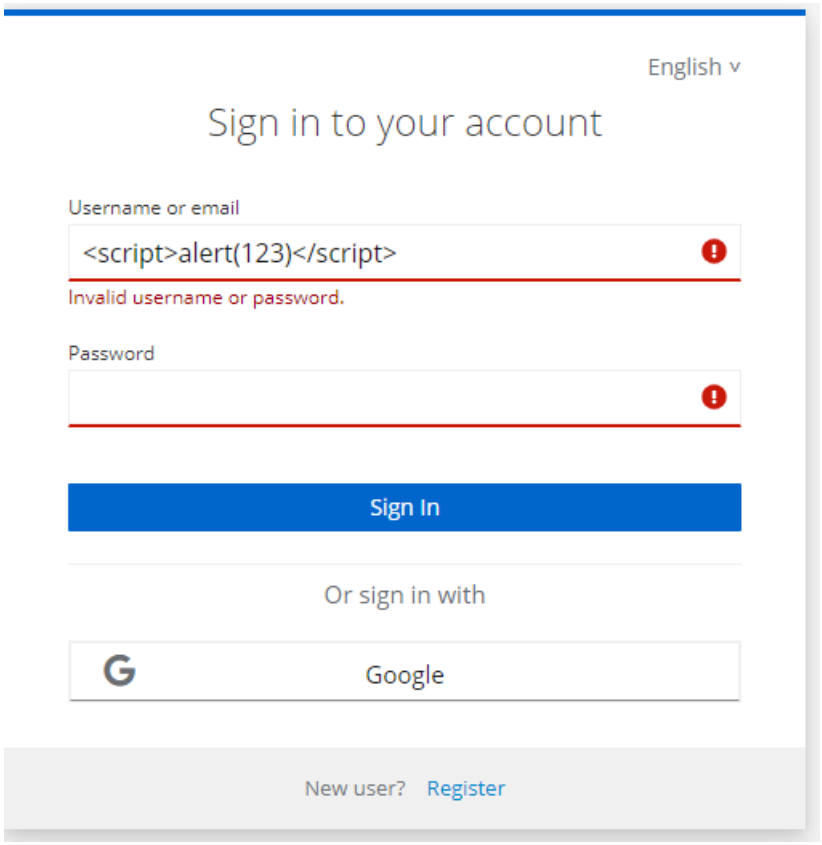
Listing 11 Kod do ataku xss

```
<script>alert(123)</script>
```



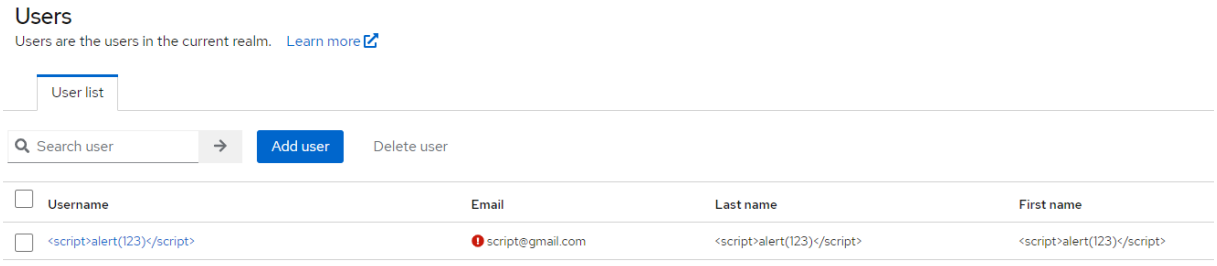
Rys. 47 Atak XSS, wywołanie alertu w przeglądarce ofiary

W formularzu logowania się do aplikacji w polu login wklejono kod z *Listing 11* w celu sprawdzenia czy strona jest podatna na atak. Aplikacja nie zgłosiła żadnego błędu, ani ostrzeżenia, że nazwa użytkownika zawiera kod java script. Keycloak nie wspiera aktywnej obrony przed wstrzykiwaniem kodu. Wciśnięcie przycisku zaloguj powoduje przetworzenie żądania (rysunek 48).



Rys. 48 Rezultat ataku XSS na formularz logowania

Nie udało się wyświetlić alertu w przeglądarce, więc formularz nie jest podatny na ten rodzaj ataku. Administrator systemu ma dostęp do kont użytkowników, może przykładowo zresetować hasło lub zablokować konto użytkownika. Można wykorzystać ten fakt do stworzenia użytkownika, który będzie miał w nazwie kod z *Listing 11*. Przeglądarka administratora po wyświetleniu listy użytkowników wykona kod jeżeli system jest podatny na ataki typu XSS (rysunek 49).



Rys. 49 Konto użytkownika z kodem JS w nazwie

Administrator systemu po wyświetleniu listy użytkowników nie zobaczył alertu w przeglądarce, a więc kod nie został wykonany. Testy wykazały, że Keycloak jest odporny na ataki typu XSS.

Wnioski

Głównym celem pracy magisterskiej było przetestowanie czy środowisko oraz funkcjonalności dostarczone przez oprogramowanie Keycloak są bezpieczne. Analizując wyniki testów automatycznych oraz manualnych można stwierdzić, że aplikacja Keycloak jest bardzo bezpieczna. Nie znaleziono znaczących podatności. Wykryto tylko nieznaczące podatności, które nie wpływają negatywnie na bezpieczeństwo aplikacji zabezpieczonej przez Keycloak. Wszystkie punkty wejścia do systemu w pełni chronią przed możliwością wstrzykiwania złośliwych skryptów. Zabezpieczony system również działa bardzo płynnie, nie wykryto spadków wydajności przy korzystaniu z zabezpieczonego oprogramowania. Podsumowując niniejszą pracę magisterską, system jest w pełni bezpieczny od znanych podatności. Autor tej pracy wybrał mechanizm autoryzacji i uwierzytelnienia jakim jest Keycloak do własnego startupu.

Spis rysunków

Rys. 1 Architektura mikroserwisów 6
Rys. 2 Architektura stworzonego rozwiązania na potrzeby testów. 11
Rys. 3 Model MVC. 11
Rys. 4 Diagram sekwencji autoryzacji do zasobu dla JWT 21
Rys. 5: Diagram sekwencji autoryzacji do zasobu dla Session 22
Rys. 6 Projekt infrastruktury 26
Rys. 7 Pobieranie tokenu z ciasteczka 27
Rys. 8 Widok formularza do logowania 29
Rys. 9 Widok ustawień klienta 30
Rys. 10 Detale stworzonego użytkownika 31
Rys. 11 Udane Logowanie na konto użytkownika 32
Rys. 12 Domyślna konfiguracja bezpieczeństwa dla Keycloak 34
Rys. 13 Widok pierwszej analizy aplikacji 35
Rys. 14 Lista zagrożenia po usunięciu zagrożenia CSP: script-src unsafe-inline 36
Rys. 15 Lista zagrożeń po usunięciu zagrożenia CSP: style-src unsafe-inline 36
Rys. 16 Lista zagrożenia po usunięciu wszystkich zagrożeń z rodziny CSP 37
Rys. 17 Zagrożenia, które eksponują informacje o aplikacji 39
Rys. 18 Wynik po próbie utworzenia 4 użytkowników 40
Rys. 19 Wygenerowanie poświadczenie dla aplikacji Keycloak 41
Rys. 20 Skonfigurowany i włączony mechanizm Recaptcha 42
Rys. 21 Widok formularza rejestracji 43
Rys. 22 Zbiór polityk, które można skonfigurować 44
Rys. 23 Aktualizacja nieaktywnego hasła 45
Rys. 24 Próba zmiany na poprzednie hasło 45
Rys. 25 Komunikat o haśle, które jest na czarnej liście 46
Rys. 26 Próba zmiany hasła na krótsze niż 8 46
Rys. 27 Brak zgodności z wyrażeniem regularnym 47
Rys. 28 Hasło takie same jak nazwa użytkownika 47
Rys. 29 Komunikat w przypadku, jeżeli hasło jest takie same jak adres email 48
Rys. 30 Komunikat o błędnej ilości znaków specjalnych w haśle 48
Rys. 31 Komunikat o błędnej ilości wielkich liter w haśle 49
Rys. 32 Komunikat o błędnej ilości małych liter w haśle 49
Rys. 33 : Komunikat o błędnej ilości cyfr w haśle 49
Rys. 34 Komunikat informujący, że hasło jest za długie 49
Rys. 35 Widok możliwych opcji do skonfigurowania w mechanizmie OTP 50
Rys. 36 Widok konfiguracji generatora haseł OTP 51
Rys. 37 Widok z aplikacji Microsoft Authenticator 52
Rys. 38 Widok okna mechanizmu walidacji OTP 52
Rys. 39 Domyśla konfiguracja wykrywania Brute force 53
Rys. 40 Konfiguracja słownika do łamania hasła 54
Rys. 41 Wynik ataku Brute force 54
Rys. 42 Komunikat po próbie logowania na konto, które zostało zablokowane na stałe 55
Rys. 43 Rezultat ręcznego ataku SQL injection 56
Rys. 44 Treść przechwyconego żądania 57
Rys. 45 Wywołanie testów sqlmap 57
Rys. 46 Rezultat nieudanego ataku SQL injection na aplikacje 58
Rys. 47 Atak XSS, wywołanie alertu w przeglądarce ofiary 59

RYS. 48 REZULTAT ATAKU XSS NA FORMULARZ LOGOWANIA 60
RYS. 49 KONTO UŻYTKOWNIKA Z KODEM JS W NAZWIE 60

Spis listingów

Listing 1 Kontrakt pomiędzy dwoma interfejsami 7
Listing 2 Token JWT wygenerowany przez Keycloak podczas logowania 19
Listing 3 Konfiguracja docker compose 26
Listing 4 Konfiguracja dostępu do zasobu 27
Listing 5 Filtr przekierowujący użytkownika na stronę logowania w frontend 28
Listing 6 Rozszyfrowany zawartości tokenu JWT stworzony przez Keycloak 32
Listing 7 Domyślna konfiguracja dla CSP script 35
Listing 8 Konfiguracja która niweluje zagrożenie Wildcard Directive 36
Listing 9 Ostateczna konfiguracja, która w pełni niweluje podatności z rodziny CSP 37
Listing 10 Teść żądań przesyłanych do Keycloak podczas logowania 38
Listing 11 Kod do ataku xss 59

Bibliografia

[1] „Wojska Obrony Cyberprzestrzeni,” [Online]. Available: https://www.cyber.mil.pl. [Data uzyskania dostępu: 24 04 2023].

[2] „Open source,” [Online]. Available: https://opensource.org. [Data uzyskania dostępu: 24 04 2023].

[3] „Keycloak,” [Online]. Available: https://www.keycloak.org. [Data uzyskania dostępu: 29 04 2023].

[4] D. Stuttard, The Web Application Hacker’s Handbook, 2007.

[5] „Spring Boot,” [Online]. Available: https://spring.io. [Data uzyskania dostępu: 29 04 2023].

[6] „VueJs,” [Online]. Available: https://vuejs.org. [Data uzyskania dostępu: 29 04 2023].

[7] „Docker,” [Online]. Available: https://www.docker.com. [Data uzyskania dostępu: 29 04 2023].

[8] „OWASP ZAP,” [Online]. Available: https://www.zaproxy.org. [Data uzyskania dostępu: 24 04 2023].

[9] M. Fowler, Patterns of Enterprise Application Architecture, 2002.

[10] I. Ristic, Modsecurity Handbook, 2010.

[11] „Pure cloud solutions,” [Online]. Available: https://www.purecloudsolutions.co.uk/how-long-will-it-take-to-hack-your-password. [Data uzyskania dostępu: 06 05 2023].

[12] „Nord Pass,” [Online]. Available: https://nordpass.com. [Data uzyskania dostępu: 06 05 2023].

[13] O. Goldreich, Foundations of Cryptography: Volume 1, 2007.

[14] „Spring Data JPA,” [Online]. Available: https://docs.spring.io/spring-data/jpa/docs/current/reference/html. [Data uzyskania dostępu: 08 05 2023].

[15] S. Fogie i J. Grossman:, XSS Exploits Cross Site Scripting Attacks and Defense, 2007.

[16] „Wireshark,” [Online]. Available: https://www.wireshark.org. [Data uzyskania dostępu: 02 06 2023].

[17] „Burp Suite,” [Online]. Available: https://portswigger.net/burp. [Data uzyskania dostępu: 16 05 2023].

[18] „Python,” [Online]. Available: https://www.python.org. [Data uzyskania dostępu: 16 05 2023].

[19] „Postman,” [Online]. Available: https://www.postman.com. [Data uzyskania dostępu: 02 06 2023].

[20] „JWT,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519. [Data uzyskania dostępu: 02 05 2023].

[21] „HTTP,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2616. [Data uzyskania dostępu: 02 05 2023].

[22] M. Sikorski i A. Honig, Practical Malware Analysis, 2012.

[23] D. Graham, Etyczny haking. Praktyczne wprowadzenie do hakingu, 2022.