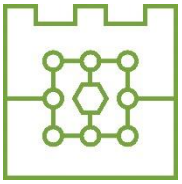
Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji



Anna Dybel
148901

Analiza bezpieczeństwa aplikacji Next.js zintegrowanej z platformą Auth0

Security analysis of the Next.js application integrated with Auth0

Praca magisterska
na kierunku Informatyka
specjalność Cyberbezpieczeństwo

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Kraków, 2024

Spis treści

Wykaz oznaczeń

Wykaz ważniejszych oznaczeń i skrótów

Skrót	Rozwinięcie skrótu	Tłumaczenie terminu na j. Polski
API	Application Programming Interface	Interfejs programowania aplikacji
CLI	Command Line Interface	Interfejs wiersza poleceń
CSS	Cascading Style Sheets	Kaskadowe arkusze stylów
CSP	Content Security Policy	Polityka bezpieczeństwa treści
CSR	Client-side rendering	Generowanie po stronie klienta
CSRF	Cross-Site Request Forgery	Fałszowanie żądań między witrynami
DDoS	Distributed Denial of Service	Rozproszona odmowa usługi
DOM	Document Object Model	Obiektowy model dokumentu
DoS	Denial of Service	Odmowa usługi
HSTS	HTTP Strict Transport Security	Ścisłe bezpieczeństwo transportu HTTP
HTTP	HyperText Transfer Protocol	Protokół przesyłania dokumentów hipertekstowych
HTTPS	HyperText Transfer Protocol Secure	Szyfrowany protokół przesyłania dokumentów hipertekstowych
ICT	Information and Communications Technology	Technologie informacyjno-komunikacyjne
IP	Internet Protocol	Protokół internetowy
JSON	JavaScript Object Notation	Notacja obiektu JavaScript
JSX	JavaScript Syntax eXtension	Rozszerzenie składni JavaScript
JWT	JSON Web Token	Sieciowy token JSON
MIME	Multipurpose Internet Mail Extensions	Uniwersalne rozszerzenia poczty internetowej
MITM	Man-in-the-Middle	Człowiek pośrodku
mXSS	Mutation-Based Cross-Site Scripting	Skrypty między witrynami oparte na mutacji
OWASP	The Open Worldwide Application Security Project	Otwarty ogólnoświatowy projekt bezpieczeństwa aplikacji

SAST	Static application security testing	Statyczne testy bezpieczeństwa aplikacji
SDK	Software development kit	Zestaw narzędzi dla programistów
SSO	Single sign-on	Pojedyncze logowanie
SQL	Structured Query Language	Strukturalny język zapytań
SSR	Server-side rendering	Generowanie stron po stronie serwera
RTT	Round-trip time	Czas podróży w obie strony
XSS	Cross-Site Scripting	Skrypty między witrynami
WWW	World Wide Web	Ogólnoświatowa sieć komputerowa
UI	User Interface	Interfejs użytkownika
URI	Uniform Resource Identifier	Jednolity identyfikator zasobu
URL	Uniform Resource Locator	Jednolity lokalizator zasobów

1. Wstęp

1.1 Motywacja

W dzisiejszych czasach rozwój technologii postępuję w bardzo szybkim tempie. Wprowadzone są coraz to nowsze i zaawansowane rozwiązania. Praktycznie wszystkie usługi zaczynają być oferowane w postaci aplikacji webowych w celu ich automatyzacji oraz wygody użytkowników. Jednak to także towarzyszy rozwojowi cyberprzestępczości. Wobec tego oferowane aplikacje powinny uwzględniać to zagrożenie i starać się minimalizować wszelkie wektory ataków. Obecnie także istnieje wiele różnych rozwiązań usprawniających proces tworzenia aplikacji webowych takich jak liczne biblioteki, platformy programistyczne czy też gotowe usługi dostarczające rozwiązania z zakresu uwierzytelniania. Bardzo ważna jest wiedza o tym co dają na takie rozwiązania w zakresie bezpieczeństwa, aby maksymalnie wykorzystać ich mocne strony oraz zadbać o te punkty, które są ich słabościami.

1.2 Cel pracy

Niniejszej **praca ma na celu przeprowadzenie analizy pod kątem bezpieczeństwa narzędzia do tworzenia aplikacji webowych jakim jest Next.js**, który bazuje na bardzo popularnej bibliotece React.js **oraz platformy Auth0 oferującej rozwiązania z zakresu uwierzytelniania i autoryzacji.** Analiza skupia się na aspektach bezpieczeństwa jakie dostarcza nam samo narzędzie oparte na React.js oraz platforma Auth0 w bezpłatnej wersji, które powinny zostać uwzględnione przy tworzeniu każdej aplikacji webowej tworzonej o te rozwiązania, aby ograniczyć luki, które sprzyjają popularnym atakom jak m. in. XSS, CSRF, DDoS, wstrzykiwaniu SQL, atakom siłowym.

1.3 Zakres pracy

Niniejsza praca składa się z części teoretycznej, która wprowadza z zagadnienia niezbędne do rozumienia tematyki pracy oraz części praktycznej pokazującej wykonane czynności i wnioski.

Część teoretyczna zawiera m. in. ogólny opis problemów i wyzwań związanych z cyberbezpieczeństwa w dzisiejszych czasach, wprowadzeniem do technologii React.js, Next.js oraz platformy Auth0. Kolejne rozdziały skupiają się na aspektach

bezpieczeństwa, czyli przeglądzie najbardziej popularnych zagrożeń, na które są narażone aplikacje webowe, rozwiązań minimalizujących powierzchnie ataków oraz metod testowania i przykładów narzędzi, które są wykorzystywane do testowania bezpieczeństwa.

Część praktyczna to m. in. analiza funkcjonalności platformy Auth0 jakie oferuję w celu ochrony przez takimi atakami jak XSS, wstrzykiwanie SQL, CSRF, siłowymi, MITM oraz innymi podatnościami, które mogą zostać wykorzystane jako punkt ataku. Następnie część praktyczna skupia się na aspektach bezpieczeństwa jakie oferuje React.js i Next.js głównie w ochronie strony klienta przez takimi atakami jak XSS, CSRF.

1.4 Metodyka pracy

Ze względów wykorzystania rozwiązania oferowanego przez stronę trzecią jakim jest platforma Auth0, przeprowadzona analiza bezpieczeństwa miała charakter testów szarej skrzynki. Testy mające na celu analizę bezpieczeństwa funkcjonalności oferowanych przez Auth0 oraz mechanizmów wspomagających tworzenie bezpiecznej aplikacji jakie oferują React.js i Next.js były wykonane przy wykorzystaniu narzędzi wspomagających testowanie bezpieczeństwa jak i też testów manualnych oraz wsparte wskazówkami i zaleceniami OWASP.

Aby móc przeprowadzić analizę bezpieczeństwa, wyżej wymienione działania musiały zostać poprzedzoną utworzeniem konta w platformie Auth0 i początkowym skonfigurowaniem usługi. Następnie została stworzona prosta aplikacja webowa Next.js, która została zintegrowana z usługą oferowaną przez Auth0.

CZĘŚĆ TEORETYCZNA

2. Problemy i wyzwania związane z bezpieczeństwem

Rozwój nowoczesnych technologii niewątpliwie przyczynił się do pozytywnych zmian i postępu we wszystkich obszarach życia człowieka. Niestety oprócz korzyści, które niosą te zmiany, także wzrosło zagrożenie bezpieczeństwa, na które jest narażony każdy korzystający z nowoczesnych urządzeń i Internetu. Wobec tego w ostatnim czasie nastąpił także rozwój obszaru cyberbezpieczeństwa, którego głównym celem jest dbanie o wysoki poziom bezpieczeństwa jak i szybkie reakcje na ulepszanie metod obrony przed coraz to bardziej wyrafinowanymi atakami ze strony cyberprzestępców.

Głównym celem przeprowadzania ataków cybernetycznych, w dużym uogólnieniu, jest naruszenie poufności, integralności i dostępności danych. Skutki jakie mogą spowodować skutecznie przeprowadzone ataki to m. in. utrata danych, zakłócenia w dostarczaniu usług, uszkodzenia sprzętu, straty finansowe jak i też wizualne [1]. Motywacje atakujących do podjęcia ataku mogą być różne i nie zawsze będzie to tylko kradzież cennych danych, ale także celowe spowodowanie strat finansowych lub uszczerbku na reputacji firm świadczących różne usługi.

Wiedza na temat cyberbezpieczeństwa, zagrożeń i metod zabezpieczenia się przed nimi jest nie tylko bardzo kluczową sprawą dla specjalistów ze środowiska ICT, ale także dla każdego, kto codziennie jest narażony na zostanie ofiarą potencjalnego ataku hakerskiego.

2.1 Hakerzy dziś a kiedyś

Hakerzy w dzisiejszych czasach zyskali duży rozgłos ze względu na ogromny wzrost ataków jaki nastąpił na przestrzeni ostatnich lat, który był spowodowany rozwojem technologii i jej popularyzacji. Jednak patrząc się w historię XX wieku można tam dopatrzyć się działań podobnych do działań hakerskich takich jak np. złamanie kodu Enigmy [2], w którym kluczową rolę odegrali polscy matematycy, którzy na podstawie badań maszyny byli wstanie doprowadzić do odszyfrowania tajnych wiadomości. Mimo ogromnego sukcesu rozwiązanie to miało poważną wadę, ponieważ niebyło skalowane a maszyna była ciągle udoskonalana. Jednak ich badania pozwoliły angielskiemu matematykowi Alanowi Turingowi opracować bardziej skuteczną metodą, która bez wątpienia doprowadziła do tego, że kod Enigmy nie był już bezpieczny bez względu na skomplikowanie maszyny. Ten przykład pokazuję jak duży

wysiłek należało podjąć oraz ile lat musiało upłynąć, aby moc doprowadzić do sukcesu złamania kodu [*2*]. Istnieje też wiele innych mniej lub bardziej znanych działań o podobnym charakterze jednak ze względu na to, że dawniej dostęp do technologii był mocno ograniczony jak i wiedza na ich temat, dlatego wtedy nie było to tak odczuwalne przez społeczeństwa jak teraz. Wraz z upływem lat zmieniły się cele hakerów, dziś już nie tylko duże korporacje są celem, ale także zwykli ludzie. Tak samo metody ataków stają się coraz bardziej podstępne. Hakerzy nie tylko skupiają się na aspektach sprzętowych czy lukach w kodzie mogących doprowadzić do pomyślnego ataku, ale także stosują różnego rodzaju psychologiczne sztuczki mające na celu oszukać potencjalną ofiarę. Warto także dodać, że mimo, że hakerzy kojarzą się z przestępstwem to istnieją ludzie, którzy w dzisiejszych czasach zajmują się tym zawodowo np. w bankowości m.in. w celu szukania luk w rozwiązaniach, aby eliminować wszelkie podatności bez narażania na nie użytkowników.

2.2 Od statycznych stron internetowych do rozbudowanych aplikacji webowych

W czasach, gdy sieć WWW dopiero rozwijała się i zyskiwała na popularności, strony internetowe były prostymi dokumentami HTML, które tylko dostarczały wiedzę w postaci treści i obrazków a głównie na taki były narażone sieci i serwery. Wraz z rozwojem sieci WWW i technologii służących do budowy stron internetowych, strony przestały już być statycznymi dokumentami, które można tylko przeglądać i czytać, a zaczęły implementować funkcjonalności pozwalające użytkownikom na wymianę danych z serwerem. Statyczne strony internetowe ewaluowały do serwisów zezwalających na prostą interakcję z użytkownikiem aż do rozbudowanych aplikacji webowych. Ewolucja sieci dała atakującym inną przestrzeń ataków, której celem stali się użytkownicy, którzy są niestety często bezbronni w obliczu dobrze przeprowadzonego ataku [3].

Początkowo atakujący byli skupieni na słabościach przeglądarek i protokołów, które nie były projektowane z myślą o bezpieczeństwie. Jednak z czasem przeglądarki jak i protokoły sieciowe zaczęły stawać się coraz bardziej zaawansowane pod względem bezpieczeństwa co w znacznym stopniu utrudnia przeprowadzenie skutecznego ataku. Wobec tego, dzisiejszy atakujący w dużym stopniu skupiają się na

poszukiwaniu luk w logice aplikacji m. in. spowodowanych błędami programistów, które mogą być ciężkie do znalezienia w bardzo rozbudowanych aplikacjach, ale nie niemożliwe do odkrycia przez skrupulatnych cyberprzestępców [2]. Według statystyk w latach 2020-2021 aż 17% ataków było skoncentrowanych na lukach w aplikacjach webowych a 72% luk wynikało z błędów w kodzie [4]. Dlatego bardzo istotne jest rozumienie wykorzystywanej technologii pod względem potencjalnych zagrożeń i mechanizmów minimalizujące ich wystąpienie. Nawet mała część kodu, która będzie działać poprawnie i spełniać wszystkie wymagania, ale zostanie napisana z pominięciem wiedzy dotyczącej potencjalnych zagrożeń, może być poważną luką w bezpieczeństwie, która wykryta przez atakującego może doprowadzić do ogromnych szkód.

3. Nowoczesne aplikacje webowe

Początkowo twórcy stron internetowych mieli do dyspozycji tylko HTML, CSS i JavaScript. Strony internetowe niebyły zbyt rozbudowane a ich rozwój był zdecydowanie utrudniony poprzez dostosowanie działania ich pod różne przeglądarki, które różniły się między sobą, ponieważ były rozwijane przez różne grupy programistów. W odpowiedzi na ten problem powstała biblioteka jQuery z łatwym API, która ujednoliciła prace programistów i ułatwiła tworzenie stron, które mogły działać na każdych przeglądarkach bez względu na ich różnice. To pozwoliło na znaczący rozwój stron internetowych w rozbudowane aplikacje webowe, ale i także rozwój nowych narzędzi do ich tworzeni, które stają coraz bardziej zaawansowane i abstrakcyjne dla programistów, którzy z nich tylko korzystają. Często programiści korzystają z gotowych rozwiązań, skupiając się tylko na ich wyniku a nie rozumieniu ich mechanizmów, słabych i mocnych stron. Takie podejście może bezpośrednio przyczynić do otwarcia potencjalnej powierzchni ataku.

3.1 Paradygmaty programowania w języku JavaScript

JavaScript jest językiem otwartym na programowanie oparte na różnych stylach m. in. takich jak obiektowe, proceduralne, prototypowe, funkcjonalne. Co więcej jest tak elastyczny, że pozwala także na łączenie wielu paradygmatów. Jednak najbardziej popularnymi są:

- **imperatywny**, który głównie skupia się na tym jak coś ma działać [5],
- **deklaratywny**, który skupia się na rezultacie [5].

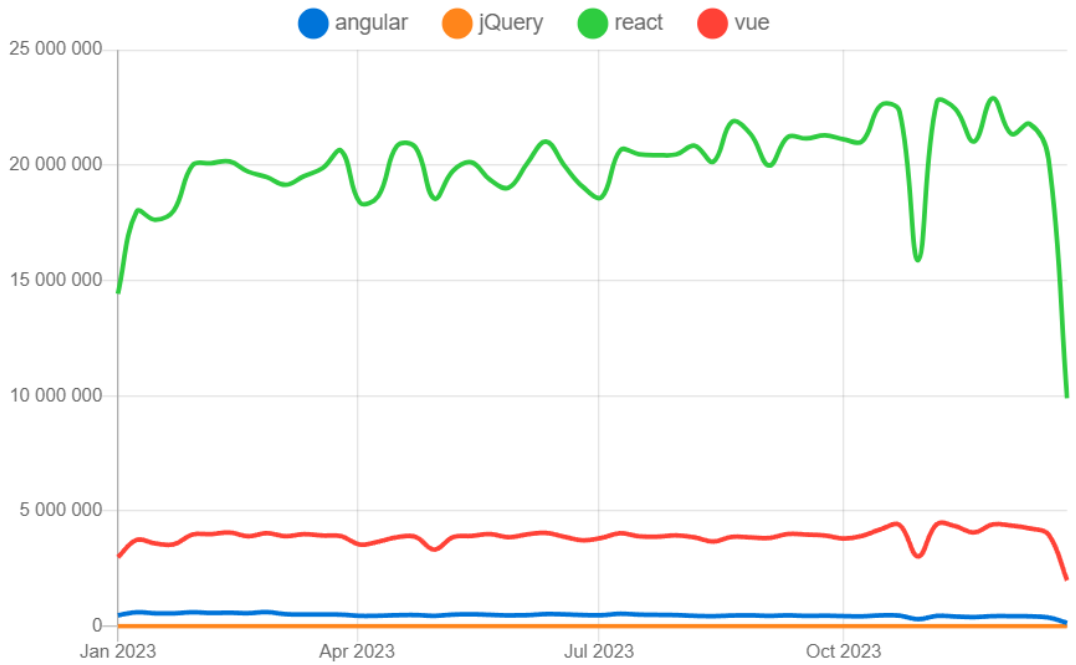
Innymi słowami podejście imperatywne polega na napisaniu sekwencji instrukcji, które mają zostać wykonane w określonej kolejności. Taki styl charakteryzuje tworzenie stron przy wykorzystaniu czystego języka JavaScript lub biblioteki jQuery. Jest on bardzo intuicyjny i prosty jednak przy rosnącej złożoności aplikacji webowych, kod jest bardzo trudny w utrzymaniu i dalszym rozwoju a to przekłada się na duże ryzyko błędów. Natomiast styl deklaratywny jest znacznie uproszczony, ponieważ programista nie skupia się na tworzeniu instrukcji w celu wykonania określonych kroków, tylko na zdefiniowaniu wyniku a następnie rozwiązania, które je zwróci. To podejście charakteryzuję nowoczesne biblioteki i platformy programistyczne do tworzenia

aplikacji webowych m. in. React.js. Niewątpliwą zaletą paradygmatu deklaratywnego jest tworzenie znacznie czystszego kodu, łatwego w rozumieniu. Ta zaleta przekłada się na uproszczenie pracy programistów jak i łatwiejsze utrzymanie i rozbudowę dużych aplikacji. Jednak istnieją też wady i jedną z nich jest trudność związana z nauką tego stylu, który w dużym uproszczeniu jest bardziej abstrakcyjny. To może prowadzić do tego, że programista może niezbyt dobrze rozumieć wykorzystywane narzędzie a to może przyczynić się do nie efektywnego tworzenia aplikacji, licznych błędów jak i luk w bezpieczeństwie [5].

3.2 Wprowadzenie do React.js i Next.js

React.js (w uproszczonej formie React) [6] jest biblioteką języka JavaScript opracowaną przez firmę Meta w 2013 roku, która do dziś jest jedną z najpopularniejszych technologii webowych wybieraną przez programistów (Rys. 1). Ten trend jest znaczącym wskaźnikiem pokazującym jak istotne jest zapewnienie odpowiednich mechanizmów obronnych a także poszerzenie wiedzy z zakresu bezpieczeństwa aplikacji tworzonych z wykorzystaniem React, zwłaszcza, że technologia jest wykorzystywana przez różne grupy programistów – zarówno specjalistów jak i początkujących programistów.

Główna idea tworzenia aplikacji webowych z wykorzystaniem React, polega na pisaniu małych komponentów, możliwych do ponownego użycia, które są kawałkiem UI a następnie łączeniu tych komponentów w coraz to większe elementy aplikacji tak jak klocki. To podejście dzieli kod na niezależne części, które są łatwiejsze w utrzymaniu i testowaniu a dodatkowo mogą być wykorzystane w wielu miejscach co znacznie zmniejsza wielkość kodu.



Rys. 1. Trend wykorzystania bibliotek i frameworków JavaScript według NPM
Źródło: [7]

Next.js [8] jest natomiast lekkim rozwiązaniem o charakterze otwartego oprogramowania stworzonym przez firmę Vercel. Istotnym faktem jest to, że Next.js jest zbudowanym na bazie biblioteki React, więc rozwiązanie to oferuje wszystko to co React a dodatkowo rozszerza go o m. in. obsługę tras i wywołania API, które nie są wbudowane w React [9]. Co więcej jest to jeden z popularniejszych frameworków, ale nie aż tak bardzo popularny jak sama biblioteka React. To także powinno być istotnym faktem, który powinien przełożyć się na wzrost bezpieczeństwa zarówno po stronie rozwijanych rozwiązanie jak również tych co z niego korzystają.

3.2.1 JavaScript i HTML w jednym pliku

Fundamentalną cechą biblioteki React jest możliwość pisania kodu JavaScript wraz z HTML dzięki składni JSX. Oczywiście nie jest to nowy język programowania tylko rozszerzenie JavaScript, który w trakcie wykonania jest tłumaczone na JavaScript. Jest to rozwiązanie, które znacznie ułatwia pisanie komponentów niż w pierwotny sposób poprzez użycie „React.createEelement”, który bardzo komplikuje pisanie nawet bardzo małych komponentów. JSX sprawia, że kod jest łatwiejszy w pisaniu, rozumieniu i utrzymaniu a także bezpieczniejszy [10].

3.2.2 Wirtualny DOM

DOM jest wewnętrzną reprezentacją strony internetowej w przeglądarce, dzięki której JavaScript ma dostęp do dokumentu HTML. Reprezentacja ta jest strukturą drzewiastą, której każdy węzeł ma właściwości i metody, które można wykorzystać do modyfikacji DOM. Pojedyncze modyfikację DOM są z reguły lekkimi operacjami, jednak w przypadku większości nowoczesnych rozwiązań aktualizacje stanu DOM są wykonywane nieefektywnie. Modyfikowana jest ogromna ilość węzłów, które często nie uległy żadnej zmianie. To bezpośrednio przyczynia się do ogromnego spadku wydajności stron internetowych. Inżynierowie, którzy tworzyli bibliotekę React skupili się na tym problemie i wprowadzili dodatkową, pośrednią warstwę, czyli Virtual DOM [11]. Główna koncepcja zakłada stworzenie reprezentacji DOM przy użyciu JavaScript, który jest znacznie szybszy w utworzeniu niż rzeczywisty. Nowy DOM jest porównywany z poprzednim DOM, trzymanym w pamięci w celu obliczenia różnicy i następnie na rzeczywistym DOM są dokonywane tylko minimalnie wymagane modyfikacje [12]. Nie jest to może tak efektywne jak modyfikacje DOM w stylu imperatywnym, ponieważ wymagane są obliczenia w celu wyznaczenia różnic, ale jest to znacznie szybsze niż w innych rozwiązaniach. To kluczowy aspekt wpływający na popularność biblioteki React.

3.2.3 Strategie generowania stron

Pierwszą ze strategii jest CRS, czyli generowanie stron po stronie klienta, które oferuje biblioteka React. Charakteryzuję się powolnym początkowym ładowaniem dużych stron. Jest to spowodowane tym, że przy żądaniu klient otrzymuje dokument HTML z początkową strukturą i minimalną liczbą znaczników oraz skryptem, który następnie przeglądarka wykonuję, aby wygenerować stronę [13]. Często temu podejściu towarzyszy pobieranie danych z serwera w postaci pliku/danych formatu JSON, ponieważ cały kod strony znajduje się w przeglądarce więc w celu aktualizacji strony o dane, które są na serwerze, muszą z niego zostać pobrane. Może to stanowić zagrożenie bezpieczeństwa ze względu na to, że często wysłane dane użytkownikowi nie są ograniczone do niezbędnych – a często wśród danych mogą znaleźć informacje poufne, które nigdy nie powinny zostać udostępnione.

Innym podejściem jest SSR, czyli generowanie po stronie serwera, które oferuje Next.js. W tym przypadku do przeglądarki trafia już całkowicie wygenerowana strona.

Towarzyszy temu oczywiście szybsze ładowanie strony w przeglądarce. Dodatkowo użytkownikowi udostępniane są wyłącznie niezbędne dane dlatego że cały proces generowania strony odbywa się po stronie serwera więc nie ma potrzeby wysyłania zbędnych danych do użytkownika. Istnieje także wstępne generowanie stron, które oznacza, że treści są generowane z wyprzedzeniem, jeszcze przed załadowaniem aplikacji w przeglądarce [9].

3.3 Zarządzanie tożsamością i dostępem

Fundamentalną częścią prawie każdej aplikacji webowej jest zapewnienie użytkownikowi bezpiecznego dostępu do zasobów i funkcjonalności aplikacji oraz ochronę poprzez odpowiednio wdrożone rozwiązanie do zarządzania tożsamością i dostępem. Rozwiązanie to obejmuje m. in. następujące procesy [14]:

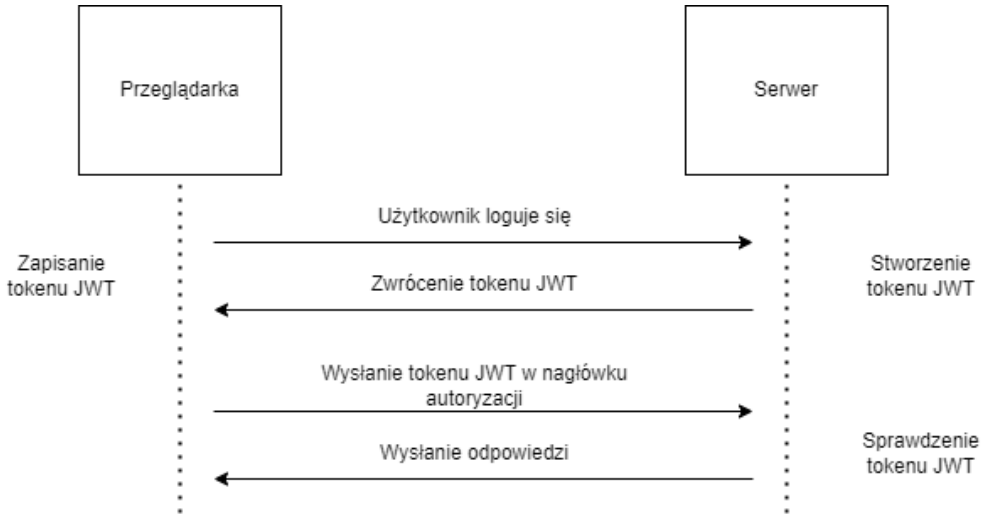
- uwierzytelnianie, czyli weryfikacja tożsamości użytkownika,
- autoryzacje, czyli proces sprawdzający do jakich zasobów ma dostęp użytkownik.

Implementacja takiego rozwiązania jest bardzo krytyczną częścią aplikacji i często ze względu na braku wiedzy w tym obszarze, ograniczony budżetu oraz małą ilość czasu wybierane są gotowe rozwiązania oferowane przez strony trzecie, które są rozwijane i utrzymywane przez specjalistów z tego obszaru [14].

3.3.1 JSON Web Token

JSON Web Token [15] w skrócie JWT jest otwartym standardem, który definiuje niezależny sposób bezpiecznego przesyłania informacji między dwoma stronami komunikacji jako obiekt JSON. Token JWT jest stosunkowo mały więc może być przysłany szybko za pośrednictwem adresu URL, parametru POST lub wewnątrz nagłówka HTTP a dodatkowo zawiera wszystkie wymagane informacje, którym można ufać, ponieważ są podpisane cyfrowo. Token JWT jest najczęściej wykorzystywany podczas procesu autoryzacji, kiedy użytkownik chce otrzymać dostęp do zasobów oraz podczas wymiany dany w celu potwierdzenia tożsamości i sprawdzenia czy dane nie zostały naruszone (Rys. 2). W tej sytuacji wraz z żądaniem wysyłany jest token JWT, na podstawie którego podejmowana jest decyzja o przyznaniu dostępu lub odrzuceniu [15]. JWT składa się z trzech następujących elementów [16]:

- nagłówek, który zawiera informacje dotyczące typu i algorytmu kryptograficznego wykorzystanego do zabezpieczenia zawartości,
- ładunek, który zawiera informacje o tożsamości użytkownika i uprawnieniach jakie posiada,
- sygnatura wykorzystywana do sprawdzenia autentyczności tokena oraz do zapewnienia, że dane nie zostały naruszone.



Rys. 2. Schemat wykorzystania tokena JWT
Źródło: opracowanie własne na podstawie [*16*]

3.3.2 Platforma Auth0

Auth0 [17] to platforma oparta na chmurze, która udostępnia szereg narzędzi, usług i bezpiecznych protokołów takich jak OAuth 2.0 do elastycznego, bezpiecznego i skalowanego tworzenia rozwiązań w zakresie uwierzytelniania i autoryzacji różnego typów aplikacji i platform. Integracja z platformą jest bardzo prosta, ponieważ Auth0 udostępnia interfejs API, biblioteki i narzędzia SDK dedykowane dla różnych docelowych technologii. Platforma pozwala na dostosowanie uwierzytelnienia w celu optymalnego bezpieczeństwa i wygody użytkowników poprzez dostosowanie opcji bezpieczeństwa i wyboru opcji logowania spośród takich jak m. in. logowanie klasyczne, uwierzytelnianie za pomocą danych biometrycznych, uwierzytelnianie wieloskładnikowe oraz możliwość pojedynczego logowanie (SSO) [17].

4. Zagrożenia bezpieczeństwa aplikacji webowych

4.1 Skrypty między witrynami (XSS)

Według OWASP najczęstszymi lukami w aplikacjach webowych są luki umożliwiające przeprowadzenie ataków XSS, które są związane z dynamicznym wykonywaniem skryptów w przeglądarkach po stronie klienta [2]. Atakujący nie potrzebuję dostępu do serwera, aby zainfekować stronę, ponieważ wystarczy, że wykorzysta funkcjonalności, które zostały udostępnione w aplikacji do wprowadzania danych z zewnątrz. Głównie narażone mogą być wszelkie formularze i dynamiczne adresy URL [19]. Rys. 3 przedstawia schemat typowego ataku XSS.

Rys. 3. Schemat typowego ataku XSS
Źródło: opracowanie własne na podstawie [19]

Ogólna charakterystyka zagrożeń i możliwości związanych z atakami XSS to m. in. [2]:

- wykonywanie skryptów nie będących częścią legalnego kodu aplikacji,
- uruchamianie złośliwego kodu bez udziału użytkownika,
- brak świadomości ze strony ofiary o tym, że nielegalny skrypt został wykonany (brak zmian w wizualizacji strony),
- przechwytywanie danych wykorzystywanych przez aplikacje jak i wysłanie danych z złośliwego serwera,
- przejmowanie kont poprzez kradzież tokena sesji.

4.1.1 Zapisane ataki XSS

Zapisane ataki XSS (ang. Stored XSS) głównie polegają na zapisie zewnętrznych danych np. testowych w bazie danych aplikacji przed wykonaniem. Są zaliczane do jednych z najbardziej niebezpiecznych ataków, ponieważ są bardzo proste do wykrycia przez atakującego a dodatkowo dają możliwość zaatakowania dużej grupy użytkowników. Jest to spowodowane tym, że dane zapisane w bazie przez jednego użytkownika mogą zostać wyświetlone innemu użytkownikowi, w najgorszym scenariuszu nawet wszyscy użytkownicy mogą paść ofiarami ataku [2].

4.1.2 Odbite ataki XSS

Odbite ataki XSS (ang. Reflected XSS) różnią się od Stored XSS tym, że zainfekowane dane nie znajdują się na serwerze aplikacji. Najczęściej tego typu atak odbywa się poprzez złośliwe łączę udostępnione ofierze np. poprzez pocztę elektroniczną. Następnie przejście pod zainfekowany adres URL, wysyła zainfekowany kod na serwer a on odsyła go do przeglądarki klienta, która interpretuje kod jako zaufany i go wykonuje [18]. Tego typu atak jest dość trudny do wykonania, ponieważ, mimo że adres URL można w łatwy sposób rozpowszechnić jednak samo nakłonienie użytkownika do przejścia pod wskazany a dodatkowo bardzo podejrzany adres może być ciężkie. Dlatego często tego typu ataki są wsparte dodatkowymi działaniami m. in. osadzeniem złośliwego adresu w formularzu [2].

4.1.3 Ataki XSS oparte na hierarchii DOM

Ataki XSS oparte na DOM (ang. DOM-based XSS) są skierowane na klienta i jego przeglądarkę poprzez modyfikacje DOM do której nie jest potrzebna interakcja z serwerem aplikacji. Są to trudne do wykonania ataki, ponieważ wymagają specjalistycznej wiedzy, jeśli chodzi o DOM i JavaScript a dodatkowo użytkownicy korzystają z wielu przeglądarek [2].

4.1.4 Ataki XSS oparte na mutacji

Mutation-based XSS jest nowym typem ataku, który polega na wykorzystaniu luki w filtrach, których zadaniem jest oczyszczenie danych w celu zapobiegania atakom

XSS. Luka może być spowodowana założeniem filtrów, które zakładają jaki wynik powinien być wygenerowany w DOM, natomiast przeglądarki mogą zastosować pewne mechanizm optymalizacyjne i przekształcić wynik w celu naprawienia struktury znaczników HTML [2].

4.2 Fałszowanie żądań między witrynami (CSRF)

Ataki typu CSRF mają na celu obejście ograniczenia dostępu do pewnych funkcjonalności jakie nakładają określone uprawnienia przez aplikacje np. wykonanie operacji w imieniu użytkownika posiadającego uprawnienia administratora. W odróżnieniu do opisanego w poprzednim rozdziale ataku XSS, do tego rodzaju ataku nie musi być wcale wykorzystany język JavaScript, ale wiedza o sposobie działania przeglądarek i odpowiednich punktów API. Dodatkowo zainfekowany kod nie musi być osadzony w legalnym kodzie, ponieważ cały atak odbywa się poprzez stronę trzecią [2]. Ogólny schemat ataku to m. in. spowodowanie, aby przeglądarka wykonała niepożądane działanie w zaufanej witrynie poprzez uruchomienie w tej samej przeglądarce złośliwej strony [18]. Warto tutaj wspomnieć, że wszelkie takie działania wykonywane są bez świadomości użytkownika, który nie widzi jakie żądania i gdzie są wysyłane. Także serwer nie jest w stanie zapobiec takim operacją, ponieważ są wykonywane w imieniu legalnego użytkownika z odpowiednimi uprawnieniami. Wobec tego są to ataki ukryte a przez to bardziej podstępne, ponieważ ani użytkownik, ani serwer może nie być świadomy o wystąpieniu ataku [2].

4.3 Wstrzykiwanie SQL

Wstrzykiwanie SQL jest jednym z najbardziej popularnych ataków polegającym na umieszczenie zapytania SQL np. w formularzu w celu zmodyfikowania zapytania po stronie serwera. Najczęstszą luką zezwalającą na tego typu atak jest stosowanie interpolacji w celu utworzenia zapytania SQL z danymi pochodzącymi z zewnątrz. Wstrzykiwanie SQL można scharakteryzować na podstawie celów jakie chce osiągnąć atakujących m. in. [18]:

- identyfikacja parametrów podatnych na wstrzykiwania,
- informacje na temat bazy danych oraz struktury tabel,
- uzyskanie poufnych informacji zapisanych w bazach,

- dodawanie i modyfikowanie danych,
- zakłócenie działania serwera,
- ominięcie uwierzytelniania,
- zmiana uprawnień.

4.4 Odmowa dostępu do usługi

Odmowa dostępu do usługi (ang. Denial of Service, DoS) jest atakiem polegającym na zalaniu serwera bardzo dużo ilością zapytań, aby go spowolnić lub całkowicie wyłączyć poprzez wykorzystanie jego zasobów w celu utrudnienia korzystania z usług zwykłym użytkownikom. Inną wersją tego ataku jest DDos (ang. Distributed Denial of Service), który charakteryzuję się generowaniem żądań przez większą liczbę zdalnych urządzeń [2].

4.5 Człowiek pośrodku (MITM)

Człowiek pośrodku (ang. Man in the middle, MITM) jest atakiem sieciowym, który następuje wtedy, gdy pomiędzy dwoma częściami komunikacji znajdzie się atakujący. W ten sposób atakujący niszczy legalny kanał komunikacyjny między klientem a serwerem i tworzy nowy kanał, który jest w pełni kontrolowany przez niego. W ten sposób atakujący dostaję dostęp do wszelkich danych wysyłanych przez użytkownika i także może nimi manipulować [20].

4.6 Przechwytywanie kliknięć

Przechwytywanie kliknięć (ang. Clickjacking) jest metodą ataku polegającą na zainicjowaniu przez użytkownika akcji, której nie zamierzał np. poprzez naciśniecie przycisku osadzonego na stronie. W celu przeprowadzenia tego ataku, atakujący nakłada legalną stronę na złośliwą poprzez załadowanie legalnej strony internetowej w ramce w taki sposób, aby złośliwe elementy były przezroczyste i niewidzialne dla użytkownika. W ten sposób naciśniecie przez użytkownika legalnego przycisku może zostać przechwycone i zainicjować całkowicie inne działanie niż chciał użytkownik [21].

4.7 Siłowe ataki

Siłowe ataki opierają się na metodzie prób i błędów w celu odgadnięcia danych uwierzytelniających użytkownika. Atakujący masowo przesyła różne kombinacje aż do momentu dostania się do konta. Atakom mogą sprzyjać słabe hasła użytkowników lub brak mechanizmów ograniczających wielokrotne próby logowania, które zakończyły się niepowodzeniem [22].

5. Metody zapobiegania atakom

Istnieje wiele różnych metod i dobrych praktyk, które mogą w znaczącym stopniu ograniczyć wystąpienie zagrożeń już na bardzo wczesnym etapie tworzenia aplikacji. W tym rozdziale zostały opisane najbardziej popularne działania mogące osiągnąć ten cel. Jednak najważniejszym aspektem w tej kwestii jest wiedza programistów w zakresie bezpieczeństwa, która nie zawsze jest na dobrym poziomie przez co często pomijane są bardzo proste a istotne luki.

5.1 OWASP

OWASP to fundacja non-profit, która m. in. dostarcza wiedzę i wskazówki dotyczące tworzenia i utrzymania aplikacji na wysokim poziomie bezpieczeństwa. Co kilka lat OWASP opracowuje bardzo popularną listę dziesięciu najważniejszych problemów bezpieczeństwa wraz z zaleceniami dotyczącymi minimalizacji tych problemów. Treści udostępniane przez OWASP mają na celu uczyć programistów i innych specjalistów obszaru ICT w tematach dotyczących bezpieczeństwa aplikacji internetowych [23].

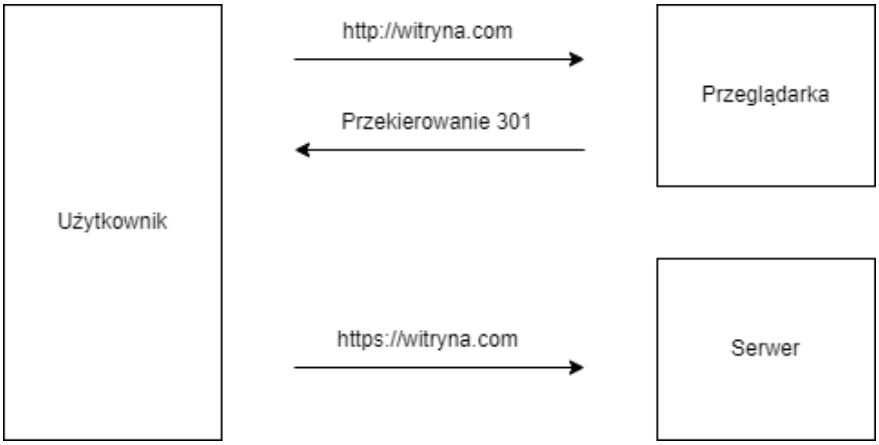
5.2 Nagłówki bezpieczeństwa protokołu HTTP

Nagłówki HTTP to zbiór reguł, które są wysyłane pomiędzy klienta a serwerem w celu określania w jaki sposób ma działać przeglądarka, aby zapewnić użytkownikowi dodatkową warstwę bezpieczeństwa. Określenie odpowiedniej konfiguracji nagłówków jest kluczowe dla każdej aplikacji, która ma na celu bezpieczeństwo. Istnieje wiele różnych mniej lub bardziej popularnych nagłówków. W następnych podrozdziałach zostały opisane najważniejsze nagłówki HTTP pod względem zapewnienia dobrych standardów bezpieczeństwa aplikacji. Warto podkreślić, że standardy często się zmieniają i różne popularne nagłówki lub ich wartości z czasem mogą stać się przestarzałe i nawet nieobsługiwane przez przeglądarki. Należy więc niebieżąco być zaznajomiony z tym temat i reagować na zmiany.

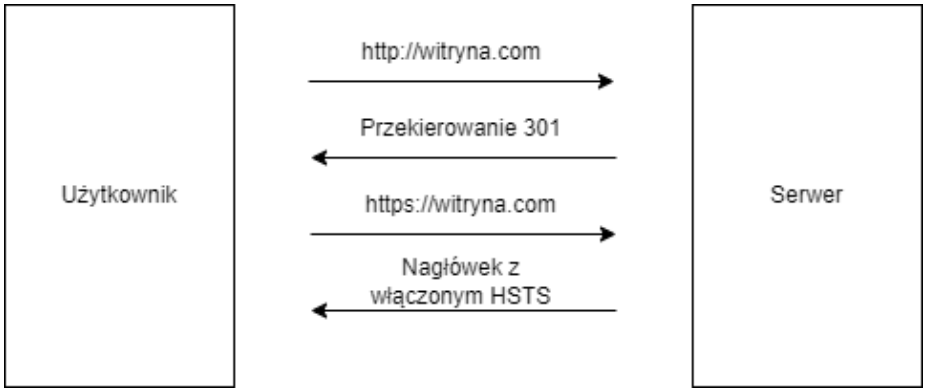
5.2.1 Ścisłe bezpieczeństwo transportu HTTP (HSTS)

Włączenie nagłówka HSTS wymusza na przeglądarce łączenie się z stroną internetową tylko poprzez szyfrowane połączenie HTTPS zamiast HTTP w celu minimalizacji

ryzyka ataku typu MITM. Wykorzystanie nagłówka HSTS gwarantuje, że użytkownik zawsze będzie korzystał z szyfrowanego połączenia nawet jeśli w przeglądarce będzie chciał połączyć za pomocą zwykłego http (Rys. 4). Jest to oczywiście gwarantowane, jeśli użytkownik w danej przeglądarce odwiedził daną stronę (Rys. 5). Aby wyeliminować ryzyko związane z pierwszym połączeniem, należy umieścić domenę na liście wstępnego ładowania zarządzaną przez Google, którą obsługują wszystkie nowoczesne przeglądarki. To rozwiązanie przyczynia się nie tylko do zwiększenia bezpieczeństwa, ale także szybszego ładowania strony [24].



Rys. 4. Połączenie ze stroną z włączonym HSTS
Źródło: opracowanie własne na podstawie [20]



Rys. 5. Pierwsze połączenie ze stroną z włączonym HSTS, której domena nie jest na liście wstępnego ładowania
Źródło: opracowanie własne na podstawie [20]

5.2.2 Polityka bezpieczeństwa treści (CSP)

Polityka bezpieczeństwa treści (ang. Content Security Policy, CSP) jest nagłówkiem, który pozwala na kontrolowanie zasobów takich jak skrypty, czcionki, obrazy czy style, z których powinna korzystać aplikacja poprzez wskazanie przeglądarce, które źródła są zaufane. Dobrze skonfigurowane zasady pozwalaj m. in. ograniczyć możliwość ataków XSS jednak nie wyeliminują w całości podatności na ataki. Dlatego bardzo ważne jest, aby postrzegać ten mechanizm jako dodatkową ochronę a nie jedyną [25].

5.2.3 Kontrola zagnieżdżania stron w ramkach

Nagłówek „X-Frame-Option" jest nagłówkiem bezpieczeństwa, który został zasugerowany przez firmę Microsoft w celu przeciwdziałania podatnością na ataki przechwytywania kliknięć. Włączenie go pozwala uniknąć możliwości osadzenia aplikacji w innych witrynach, jeśli przeglądarka, której używa klient wspiera obsługę tego nagłówka [26].

5.2.4 Ochrona przed podsłuchiwaniem zawartości

Nagłówek „X-Content-Type-Option" pozwala na rezygnację z mechanizmu przeglądarek zwanego podsłuchiwaniem MIME. Jest to mechanizm, który w przypadku, gdy np. nagłówek Content-Type nie jest ustawiony, stara się ustalić format danych, aby poprawianie go obsłużyć np. sprawdzając bajty. Takie zachowanie przeglądarki może narazić użytkowników na niepożądane działanie ze strony atakującego poprzez obsłużenie niebezpiecznego formatu pliku. Włączenie nagłówka spowoduję, że przeglądarka nie będzie próbować odpowiednio obsłużyć zasobów, które nie są zgodne z nagłówkiem Content-Type [26].

5.2.5 Zabezpieczanie plików Cookies

Jedną z metodą zabezpieczająco plik cookie, które może zawierać dane sesji jest ustawienie parametru „HttpOnly" w nagłówku Set-Cookie. Parametr ten sprawia, że plik Cookie może zostać odczytany tylko przez HTTP a to oznacza, że nie ma możliwości dostania się do niego poprzez JavaScript [27]. Innym parametrem pomagającym w utrudnieniu wykradnięcia pliku Cookie, zwłaszcza przy ataku CSRF,

jest „SameSite”. Parametr ten może mieć dwie wartości takie jak „Strict” i wtedy Cookie nie zostanie wysłany do innej domeny i trochę łagodniejszy „Lax” [28].

5.2.6 Polityka prywatności użytkowników

Nagłówek Referrer-Policy używany jest w celu kontrolowania jakie informację zostaną wysłane, jeśli użytkownik przejdzie pod jakiś adres URL umieszczony na stronie. Innymi słowami pozwala to na zapewnienie użytkownikowi prywatności poprzez ograniczenie wycieku informacji stroną trzecim [26].

5.3 Działania minimalizujące luki

W poprzednim rozdziale zostało wspomniane, że istnieje możliwość zabezpieczenia aplikacji na poziomie przeglądarek np. przez atakiem XSS poprzez skorzystanie z odpowiednich nagłówków bezpieczeństwa HTTP. Jest to oczywiście dodatkowe wsparcie w minimalizacji ryzyka wystąpienia tego ataku, które powinno towarzyszyć dobrym praktykom.

Pierwszą zasadą jaką powinien kierować się każdy twórca aplikacji webowej to m. in. traktowanie wszystkich danych, które pochodzą od użytkownika jako niezaufanych i niebezpiecznych. Oznacza to, że dane przed dalszym ich wykorzystaniem w aplikacji powinny być poddane walidacji pod względem spełniania odpowiednich kryteriów. Walidacja może być syntetyczna, która m. in. polega np. na sprawdzeniu poprawności adresu e-mail albo adresów URL czy nie zawierają szkodliwych protokołów takich jak „data” lub „javascript”. Drugim typem walidacji jest semantyczna, czyli sprawdzenie czy dane spełniają kryteria pod względem odpowiednich przedziałów np. liczby znaków. Pozwala to na wstępie przefiltrować informacje i zaakceptować tylko te, które spełniają wymagania [18].

Znaki specjalne powinny zostać odpowiednio przetłumaczone do formy, która nie jest niebezpieczna w zależności od środowiska, w którym będą używane. Tego typu proces powinien zostać wykonany przed przekazaniem danych do odpowiedniej warstwy aplikacji. Jeśli chodzi o HTML to znak „<” powinien zostać przekonwertowany do „<” natomiast „>” do „>”. Natomiast w kontekście języka JavaScript wszelkie znaki, które nie są znakami alfanumerycznymi powinny zostać zakodowane w standardzie Unicode [18].

Poddawanie danych procesowi oczyszczania może być kolejnym elementem wpływającym na ograniczenie potencjalnych luk. Polega on m. in. na usunięciu z danych zbioru znaków mogących przyczynić się do niepożądanego działania aplikacji. Często w celu wykonania tego procesu korzysta się gotowych bibliotek, które należy monitorować, ponieważ mogą zawierać luki w swoich rozwiązaniach. Przykładem takiej luki jest wcześniej opisana możliwość wykonania mXSS, która została odkryta w bibliotece służącej do oczyszczania danych.

Kolejnym aspektem wpływającym na eliminacje potencjalnych zagrożeń jest stosowanie gotowych rozwiązań. Ogólnie pisanie aplikacji całkowicie od zera może być nie rozsądne, jeśli nie dysponujemy odpowiednim czasem, budżetem i wiedzą. Dlatego wtedy warto skorzystać z gotowych usług oferowanych przez strony trzecie, które specjalizują się w danym obszarze i zapewniają wysokiej jakości rozwiązania. Kierując się wyborem należy używać rozwiązań z zaufanych źródeł, które są aktywnie utrzymywane i aktualizowane oraz powszechnie używane [18].

Oprócz wyżej wspomnianych aspektów bardzo ważne jest też to, aby udostępniać użytkownikowi tylko niezbędne funkcjonalności i dane, które są niezbędne do poprawnego działania aplikacji. W ten sposób też możemy zminimalizować powierzchnie potencjalnych ataków.

6. Testowanie bezpieczeństwa

Testowanie aplikacji pod względem bezpieczeństwa jest kluczowym aspektem każdej aplikacji w celu wyeliminowania słabych punktów, zwiększenie jakości kodu i możliwość łatwego utrzymania aplikacji. Jednym z elementów towarzyszącym tworzenie aplikacje jest robienie testów, które najczęściej mają na celu sprawdzenie funkcjonalności pod względem wymagań, które mogą często nie obejmować bezpieczeństwa ze względu na brak wiedzy w tym zakresie. Tak samo brak wiedzy może obejmować innych członków zespołu, którzy rozwijają aplikacje. Wobec tego wykonywanie testów bezpieczeństwa może zapobiec także błędom wynikającym z ograniczonej wiedzy w temacie potencjalnych luk i ogólnie bezpieczeństwa aplikacji webowych.

6.1 Testy penetracyjne

Testy penetracyjne są jedną z bardziej pomocnych technik stosowanych do testowania bezpieczeństwa. Testy wykonywane są na aplikacji z poziomu użytkownika a tester poszukuje możliwość obejścia zabezpieczeń poprzez znalezienie luk jak prawdziwy hacker stąd też często nazwy się to etycznym hakowaniem [18]. Testy penetracyjne może podzielić na trzy kategorie według informacji jakie zostaną udostępnione testerom:

- testy czarnej skrzynki (ang. Black-box) – w tym podejściu tester jest jak przeciętny haker, który ma dostęp do aplikacji poprzez adres URL, ale nie wie nic więcej, poza tym co wyświetla mu się w przeglądarce. Tester może mieć dostęp do aplikacji jako autoryzowany użytkownik lub nieautoryzowany [18]. W tym przypadku testujący musi wykazać się dużą wiedzą i umiejętnością wykorzystania różnych narzędzi umożliwiających wykonywanie ataków, ale także spostrzegawczością. Ta metoda najbardziej odwzorowuje rzeczywiste ataki, ale jest też najdroższą opcją,
- testy białej skrzynki (ang. White-box) – jest to podejście przeciwne do testów czarnej skrzynki. W tym przypadku tester ma dostęp do całego kodu źródłowego, dokumentacji, sieci i systemu. Tester skupia się na testowaniu funkcjonalności, algorytmów zależności pod kątem ich podatności na zagrożenia. Najczęściej tego rodzaju testy są wykonywane przed

wdrążeniem aplikacji. Daje to też możliwość poprawy błędów od razu oraz zwiększenie pokrycia kodu testami [18]. Ewentualną wadą tego typu testowania jest to, że tester mając pełen dostęp do wszystkiego może nie zauważyć tego co haker, który nie ma takiej wiedzy,

- testy szarej skrzynki (ang. Gray-box) - jest to kombinacja dwóch wcześniej opisanych metod. W tym podejściu tester zna funkcjonalności i wewnętrzne mechanizmy, ale ma dostęp tylko do aplikacji jako użytkownik bez dostępu do kodu źródłowego [18].

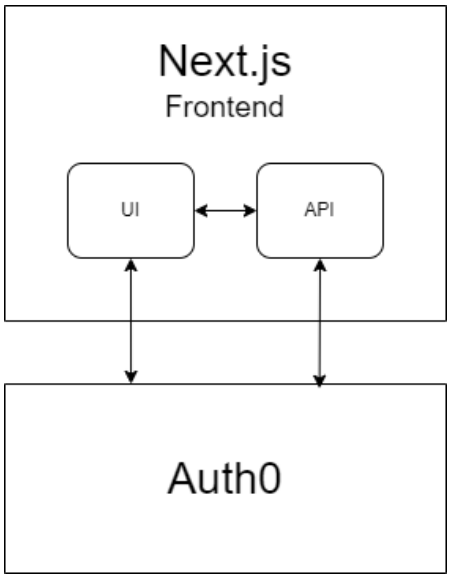
6.2 Narzędzie wspomagające testowanie bezpieczeństwa

Testowanie bezpieczeństwa można wykonać ręcznie, lecz ze względów na złożoność aplikacji oraz postęp metod hakerskich nie jest możliwe przeprowadzenie skrupulatnych testów w miarę rozsądnym czasie. Istnieje wiele płatnych i bezpłatnych narzędzi, które mają na celu zautomatyzowanie tych czynność. Jednym z rodzajów takich narzędzie są narzędzia do statycznej analizy kodu (SAST), które zaliczają się do testów metodą białej skrzynki. Ich zadaniem jest skanowania kodu źródłowego i konfiguracji w celu szukania błędów mogących przyczynić się do powstania luki bezpieczeństwa już w bardzo wczesnym etapie tworzenia aplikacji [29]. Kolejnym przydatnymi narzędziami są narzędzia do przechwytywania i modyfikowania ruchu HTTP/HTTPS w obu kierunkach przy wykorzystaniu serwera pośredniczącego. Przykładem takiego narzędzia jest Burp Proxy [18]. Innym bardzo popularnym i darmowym narzędziem jest OWASP ZAP, które dostarcza wiele różnych funkcjonalności taki jak pasywne i aktywne skanowanie wszystkich żądań i odpowiedzi HTTP/HTTPS, skanowanie portów, przeszukiwanie wszystkich linków, testowanie niebezpiecznych ładunków [30].

CZĘŚĆ PRAKTYCZNA

7. Przygotowanie i konfiguracja projektu

Praktyczna część pracy polegała na stworzeniu prostej nowoczesnej aplikacji webowej (Rys. 6) z wykorzystaniem platformy programistycznej Next.js, zintegrowanie jej z Auth0 dostarczającej rozwiązania z zakresu uwierzytelniania na podstawie, której zostały wykonane testy w kontekście bezpieczeństwa. Głównym celem testów było sprawdzenie czy platforma Auth0 jest bezpiecznym rozwiązaniem oraz jakie mechanizmy w minimalizowaniu powierzchni ataków oferuje Next.js i React.js a które aspekty programista sam musi zadbać, głównie skupiając się na środowisku użytkownika.



Rys. 6. Architektura aplikacji webowej
Źródło: opracowanie własne

7.1 Stos technologiczny i środowisko programistyczne

Niniejsza część praktyczna pracy został zrealizowana przy wykorzystaniu następujących technologii:

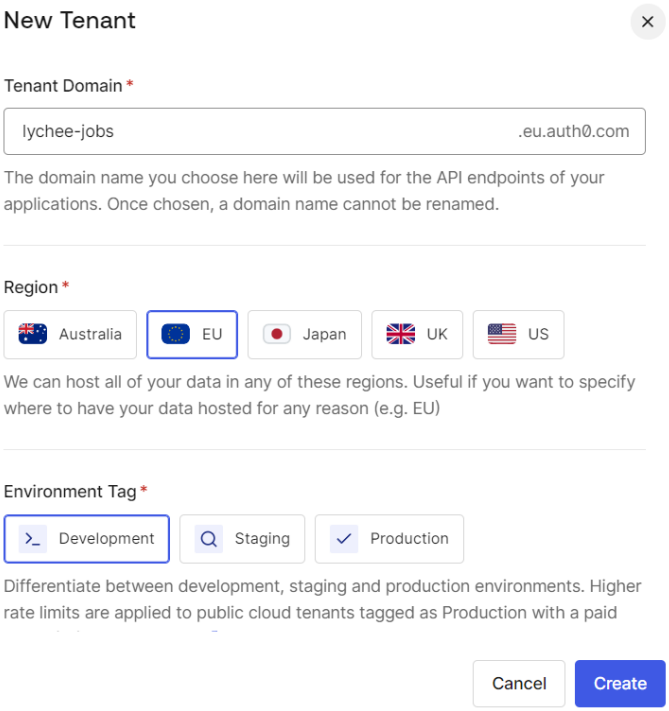
- Node.js v18.15.0 i npm v9.5.0,
- Next.js v13.5.4, React.js v18.0.0, nextjs-auth0 v3.2.0 oraz typescript v5.0.0.

Do pracy z kodem został wykorzystany darmowy edytor kodu Microsoft Visual Studio Code a do analizy bezpieczeństwa program OWASP ZAP oraz Burp Suit i Postman.

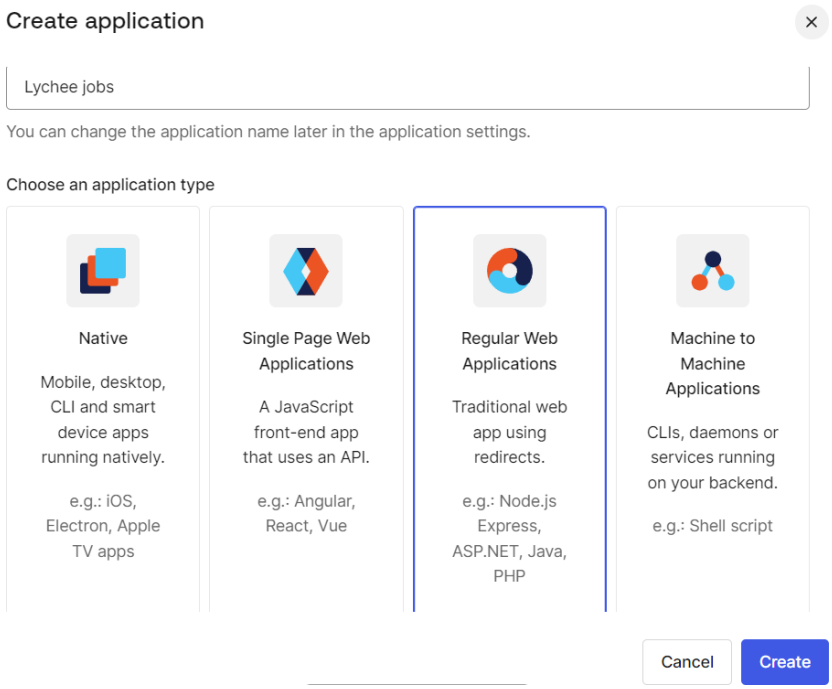
7.2 Wstępna konfiguracja

Pierwszym krokiem umożliwiającym zbudowanie aplikacji opartej o platformę programistyczną Next.js było utworzenie szkieletu aplikacji przy pomocy dedykowanego narzędzia CLI „create-next-app”, który tworzy aplikację korzystając z domyślnego szablonu z wstępną konfiguracją i zależnościami [31]. Aby zastosować się do najlepszych praktyk tworzenia nowoczesnych aplikacji webowych i aspektów bezpieczeństwa, w celu użycia create-next-app posłużono się narzędziem npx, które jest obecnie standardem, dzięki któremu zawsze korzystamy z aktualnej wersji narzędzia CLI bez jego globalnej instalacji.

Po utworzeniu szkieletu aplikacji, kolejnym krokiem było zintegrowanie aplikacji z Auth0. Aby moc skorzystać z Auth0 należało założyć darmowe konto i stworzyć nową dzierżawę (Rys. 7) wraz z dodanie do niej aplikacji typu „Regular Web Application” (Rys. 8).

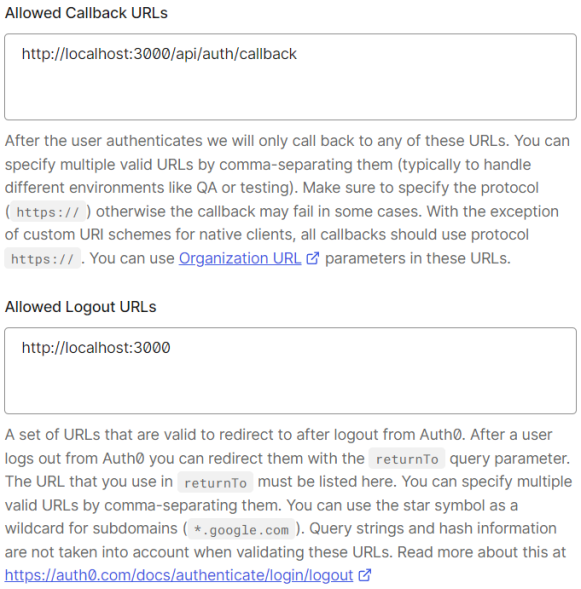


Rys. 7. Utworzenie nowej dzierżawy w Auth0
Źródło: opracowanie własne



Rys. 8. Dodanie nowej aplikacji w stworzonej dzierżawie
Źródło: opracowanie własne

Ostatnim krokiem po utworzeniu instancji było także jej skonfigurowanie poprzez określenie adresu URL na który Auth0 ma przekierować użytkownika po uwierzytelnieniu oraz po wylogowaniu (Rys. 9). Jest to bardzo ważny punkt i nie można tutaj zostawić symbolu wieloznacznego.



Rys. 9. Konfiguracja adresów przekierowania zwrotnego po uwierzytelnieniu i wylogowaniu użytkownika Źródło: opracowanie własne

Po pomyślnym stworzeniu i skonfigurowaniu instancji Auth0 należało ją zintegrować z aplikacją Next.js. W tym celu skorzystano z biblioteki nextjs-auth, która została specjalnie stworzona do implementowania uwierzytelnienia w aplikacjach Next.js i pozwala w prosty i szybki sposób zintegrować aplikację z Auth0 [32]. Aby pozwolić na komunikację aplikacji z dzierżawą należało dodać odpowiednie zmienne środowiskowe, które definiują wartości konfiguracyjne Auth0 przedstawione na Listing 1. Warto zwrócić tutaj uwagę, że sposób w jaki zmienne zostały nazwane gwarantuje, że zmienne przedstawione na Listing 1 nie będę udostępnione w środowisku klienta.

```
AUTH0_SECRET=''
AUTH0_BASE_URL='http://localhost:3000'
AUTH0_ISSUER_BASE_URL='https://lychee-jobs-dev.eu.auth0.com'
AUTH0_CLIENT_ID=''
AUTH0_CLIENT_SECRET=''
```

Listing 1. Zmienne środowiskowe potrzebne do zintegrowania Auth0 z aplikacją
Źródło: opracowanie własne na podstawie [32]

Ostatnim krokiem było dodanie po stronie serwera możliwości obsługi tras służących m. in. do logowania, wylogowania, przekierowania po pomyślnym zalogowaniu użytkownika (Listing 2).

```
import { handleAuth } from "@auth0/nextjs-auth0";

export const GET = handleAuth();
```

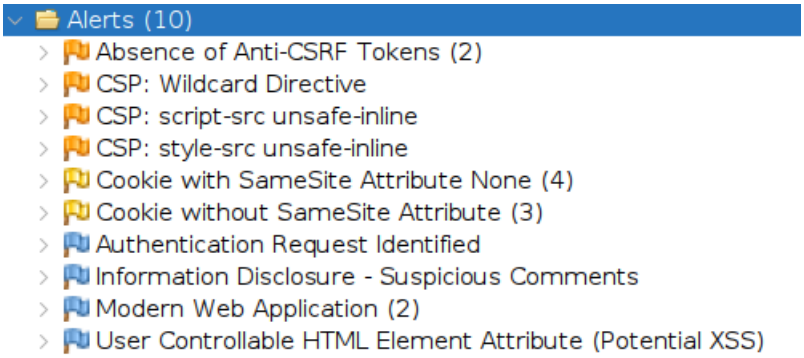
Listing 2. Stworzenie i obsługa dynamicznych tras
Źródło: opracowanie własne na podstawie [32]

W ten sposób aplikacja została w pełni zintegrowana z darmową wersją Auth0 z domyślną konfiguracją i po przejściu na odpowiednie adresy URL można korzystać z usług uwierzytelniania.

8. Analiza bezpieczeństwa platformy Auth0

8.1 Analiza wstępnej konfiguracji platformy Auth0

Rys. 10 przedstawia wyniki pasywnego skanowania wykonanego programem OWASP ZAP w postaci alertów o różnym stopniu, który sugeruje poziom możliwego zagrożenia na jakie narażona jest aplikacja. Warto tutaj zaznaczyć, że program dokonuję analizy na podstawie ruchu HTTP według ustalonych ogólnych zasad, nie znając dokładnie danego rozwiązań. Wobec tego, mimo uzyskanych alertów bezpieczeństwa należy także ocenić czy dana aplikacja webowa nie stosuję innych rozwiązań w celu uzyskania zabezpieczenia przed danymi atakami np. z poziomu kodu.



Rys. 10. Wynik pasywnego skanowania narzędziem OWASP ZAP
Źródło: opracowanie własne

Pierwszym alertem bezpieczeństwa jest informacja o braku tokenu Anti-CSRF w formularzu logowania, który jest jednym z możliwych zabezpieczeń przed atakiem CSRF, ale nie jedynym. Jego poziom priorytetu jest bardzo wysoki, ponieważ brak zabezpieczenia przed tym atakiem może narazić użytkownika na podstępne wykonanie działań, których nie będzie świadomy [18]. Z alertu można dowiedzieć się, że podczas skanowania nie znaleziono w formularzu żadnego parametru, który mógłby sugerować istnienie takiego tokenu. Oznacza to, że skanowanie opiera swoje poszukiwania o znane wzorce nazw używanych dla parametru przechowującego token i nie uwzględnia przypadku, w którym zostałaby wykorzystana niestandardowa nazwa. Według dokumentacji Auth0, platforma przeciwko atakom CSFR stosuję parametr „state”, który jest ciągiem znakowym. Badając strukturę DOM danej strony a dokładnie formularz logowania, który został zaprezentowany na Rys. 11 można zauważyć istnienie ukrytej kontrolki z nazwą „state” i wartością przechowującą długi ciąg znaków. Oznacza to, że

zabezpieczenie przez atakiem CSRF istnieje i panel logowania nie jest narażony na niebezpieczeństwo ze strony tego ataku.

```
▼<form method="POST" class="cb24974bd c337cabdf" data-form-primary="true"> == $0
  <input type="hidden" name="state" value="hKFo2SBZeW9YMnk2RERhN0xmM2p2YnBjUFBO
  Q0cwNlFjX1lBbKFur3VuaXZlcnNhbC1sb2dpbqN0aWTZIDkyWDc1dVc2bUE0Z0xVTGdPRFFqc3o5e
  U5QUTduc250o2NpZNkgZms5TEd0WEdsMUI4dEx1WXdEemNneWtvSEZFT0xHT1Y">
▶ <div style="visibility: hidden !important; position: absolute !important"
  aria-hidden="true">…</div>
▶ <div class="c711025ad c95024724">…</div>
▶ <p class="c3964fa63 c8322da11">…</p>
▶ <div class="c63db4457">…</div>
</form>
```

Rys. 11. Struktura DOM formularza logowania
Źródło: opracowanie własne

Kolejne trzy alerty dotyczą CSP, które mają na celu ograniczyć ataki typu XSS oraz Clickjacking poprzez poinstruowanie przeglądarki po stronie klienta jakie skrypty i zasoby są zaufane i mogą zostać wykorzystane w danej aplikacji webowej. Domyślnie Auth0 ustawia tylko nagłówki zapobiegające atakom metodą przechwytywania kliknięć (Rys. 12). Według dokumentacji platformy nie ma możliwość manipulacji nagłówkami CSP. Jedynym sposobem na to jest stworzenie własnej strony logowania.

```
HTTP/1.1 200 OK
Date: Sat, 25 Nov 2023 23:33:10 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
CF-Ray: 82bdb1f27a1935b8-WAW
CF-Cache-Status: DYNAMIC
Cache-Control: no-store, max-age=0, no-transform
Content-Language: en
ETag: W/"55dc-Jyj9GPQbdu/g0Mb1QNyFZlq+v7I"
Expires: Sat, 25 Nov 2023 23:33:10 GMT
Strict-Transport-Security: max-age=31536000; includeSubDomains
Vary: Accept-Encoding
Content-Security-Policy: frame-ancestors 'none'
Pragma: no-cache
Referrer-Policy: same-origin
X-Auth0-RequestId: 048b87e4b584b0d8c059
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-RateLimit-Limit: 20
X-RateLimit-Remaining: 19
X-RateLimit-Reset: 1700955197
X-Robots-Tag: noindex, nofollow
X-XSS-Protection: 1; mode=block
Server: cloudflare
alt-svc: h3=":443"; ma=86400
content-length: 21980
```

Rys. 12. Nagłówek odpowiedzi zawierający atrybut CSP z flagą zapobiegającą atakom typu przechwytywania kliknięć
Źródło: opracowanie własne

Następne dwa alerty związane są z ochroną plików Cookie przed ich kradzieżą przez złośliwe strony uruchomione w tej samej przeglądarce w ramach ataku CSRF. OWASP ZAP zgłosił ten problem, ponieważ nie wykrył atrybutu „SameSite" z wartością „Lax" lub „Strict", które zapobiegają kradzieży plików Cookie. Warto tutaj zwrócić uwagę, że jest to jedno z zabezpieczeń przed atakami CSRF a wcześniej zostało wspomniane, że Auth0 posiada taką ochronę. Dodatkowo został także użyty atrybut „secure" i „HttpOnly", który gwarantuje ze pliki Cookie będą wysyłane tylko poprzez HTTPS.

Pozostałe alerty mają charakter informacyjny i mogą dać atakującym potencjalne informację, które mogą go naprowadzić na lukę w zabezpieczeniach m. in. o tym, że witryna jest napisana z wykorzystaniem nowych technologii albo poprzez komentarz w skrypcie. Wyeliminowanie tego typu alertów jest nie możliwe z poziomu dzierżawy Auth0 lub nawet na poziomie projektowania uwierzytelniania, jeśli mamy do czynienia z alertem „Modern Web Application".

8.2 Testowanie poświadczeń przesyłanych zaszyfrowanym kanałem

Test miał na celu sprawdzenie, czy istnieje jakiś przypadek użycia aplikacji, w którym dane uwierzytelniające między klientem a serwerem są wymieniane poprzez nieszyfrowany kanał komunikacyjny.

Pierwszym krokiem była próba załadowania strony logowania przełączając protokół HTTPS na HTTP, czyli usunięcie z adresu URL listery „s".

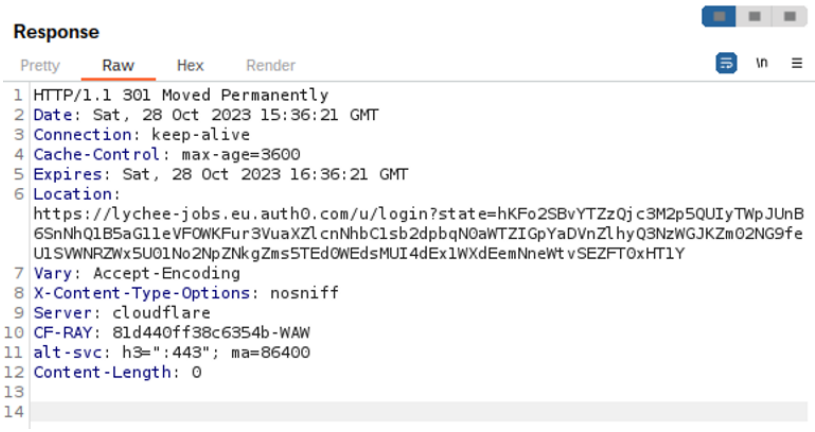
Rys. 13. Status żądania panelu logowania poprzez protokół HTTP
Źródło: opracowanie własne

Rys. 13 prezentuje status żądania podczas próby uzyskania dostępu do panelu logowania poprzez protokół HTTP, a dokładnie status przekierowania 307, który informuję przeglądarkę, że wyszukiwana treść znajduję się pod innym adresem.

Następnie przeglądarka przekierowała użytkownika na bezpieczną witrynę, mimo wymuszenia załadowania treści poprzez protokół HTTP. Stało się tak dlatego, że przeglądarka przy poprzednich wizytach otrzymała w nagłówku odpowiedzi informacje o tym, że w danej domenie został włączony mechanizm HSTS (Rys. 14). W następstwie tego, do strony logowania mamy możliwość dostępu tylko poprzez bezpieczny protokół szyfrujący. Tak samo wygląda sytuacja z dostępem do strony z formularzem do tworzenia konta jak i przypomnienia hasła, które znajdują się w tej samej domenie. Jedynym zagrożeniem dla użytkownika może być pierwsze połączenie wykonane w danej przeglądarce, która jeszcze nie wie o mechanizmie HSTS i to może spowodować lukę umożliwiającą atak MITM. Oczywiście można ten problem rozwiązać poprzez umieszczenie domeny na wstępnej liście ładowania. W tym przypadku przeglądarka jeszcze bez pierwszego połączenia z daną domeną będzie wiedzieć o włączonym mechanizmie HSTS [20].

Rys. 14. Nagłówek odpowiedzi z informacją o włączonym mechanizmie HSTS
Źródło: opracowanie własne

Następnym krokiem była próba przesłania danych poprzez protokół HTTP. W tym celu, z wykorzystaniem programu Burp Suite i serwera pośredniczącego, zostało przechwycone żądanie i zmodyfikowane tak aby dane uwierzytelniające zostały przesłane poprzez protokół HTTP.



Rys. 15. Odpowiedź po próbie przesłania danych uwierzytelniających protokołem HTTP
Źródło: opracowanie własne

Jednak przesłanie danych uwierzytelniających nie jest możliwe przy użyciu protokołu HTTP, a użytkownik po takiej próbie zostaje przekierowany do panelu logowania (Rys. 15). Również i podczas tworzenia konta jak i skorzystania z funkcji przypomnienia hasała, nie jest możliwe przesłanie żądania poprzez protokół HTTP.

8.3 Pamięć podręczna przeglądarek

Często przeglądarki implementują różne rozwiązania, które mają na celu poprawę wydajności, przyspieszenie ładowania stron oraz także usprawnić wpisanie danych przez użytkownika poprzez buforowanie danych. Jednak dane wprowadzane w celu uwierzytelnienia są danymi bardzo poufnymi i nie powinny być w żaden sposób przechowywane po stronie przeglądarek bez wyraźnej zgody użytkownika. W tym przypadku serwer powinien poinstruować przeglądarkę jak powinna się zachowywać. Analizując nagłówek odpowiedzi przedstawiony na Rys. 16 można zauważyć, że podświetlone dyrektywy tj. „Cache-Control”, i „Pragma” a ich wartości gwarantują brak wycieku poufnych informacji przez przeglądarkę. Wskutek tego próba cofnięcia się w przeglądarce do poprzedniej strony spowoduję wyświetlania się nieuzupełnionego formularza.

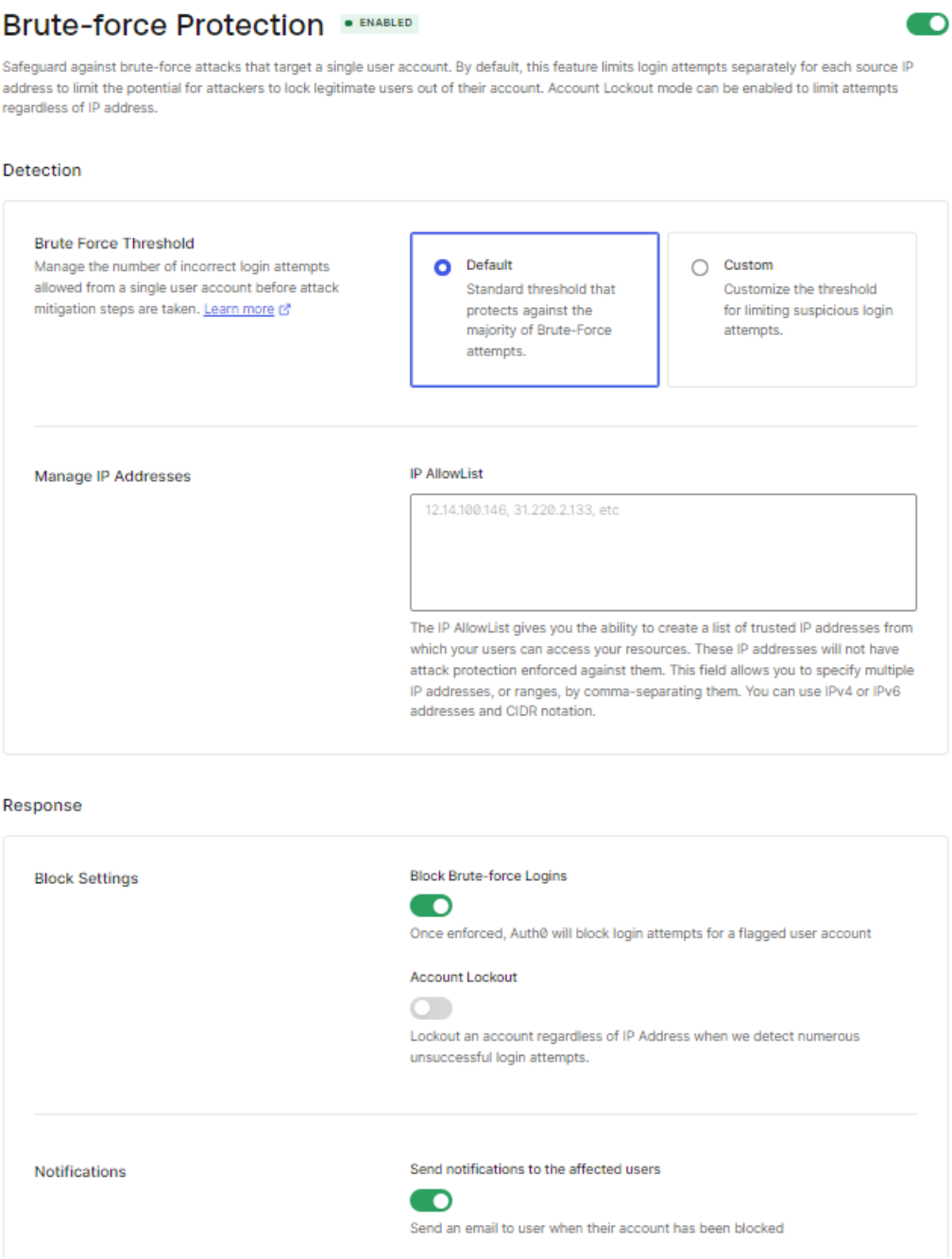
```
HTTP/1.1 200 OK
Date: Wed, 01 Nov 2023 23:18:32 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
CF-Ray: 81f7db84cb76c008-WAW
CF-Cache-Status: DYNAMIC
Cache-Control: no-store, max-age=0, no-transform
Content-Language: en
ETag: W/"4bc5-TmIWWyCsOWismeSPmXFIhns246E"
Expires: Wed, 01 Nov 2023 23:18:32 GMT
Strict-Transport-Security: max-age=31536000; includeSubDomains
Vary: Accept-Encoding
Content-Security-Policy: frame-ancestors 'none'
ot-baggage-auth0-request-id: 81f7db84cb76c008
ot-tracer-sampled: true
ot-tracer-spanid: 0a3351ee4df199c2
ot-tracer-traceid: 5a521c5858ae09af
Pragma: no-cache
Referrer-Policy: same-origin
traceparent: 00-0000000000000000005a521c5858ae09af-0a3351ee4df199c2-01
tracestate: auth0-request-id=81f7db84cb76c008,auth0=true
X-Auth0-RequestId: 93ac589e4cfaa016e2bf
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-RateLimit-Limit: 20
X-RateLimit-Remaining: 19
X-RateLimit-Reset: 1698880719
X-Robots-Tag: noindex, nofollow
X-XSS-Protection: 1; mode=block
Set-Cookie: __cf_bm=UsRVlAnExWSK4Gsh3FfofrKFy0PUMEzUW0BZVKo5oVI-1698880712-0-AcGYBitBItZbmR+
QmpNdoQxfmH8PkusVjCPmSoVdXcz7WBmZO4BXkKgHZ1iPUP1D8uJqM8SY0VcS35pNcUcFeDw=; path=/; expires=Wed, 01-Nov-23 23:48:32 GMT; domain
=.eu.auth0.com; HttpOnly; Secure; SameSite=None
Server: cloudflare
alt-svc: h3=":443"; ma=86400
content-length: 19397
```

Rys. 16. Nagłówek odpowiedzi z atrybutami zapobiegającymi buforowanie danych po stronie klienta
Źródło: opracowanie własne

8.4 Ochrona przed atakami siłowymi

Niniejszy podrozdział skupia się na analizie ochrony jaką oferuje platforma Auth0 w celu minimalizowania ataków siłowych.

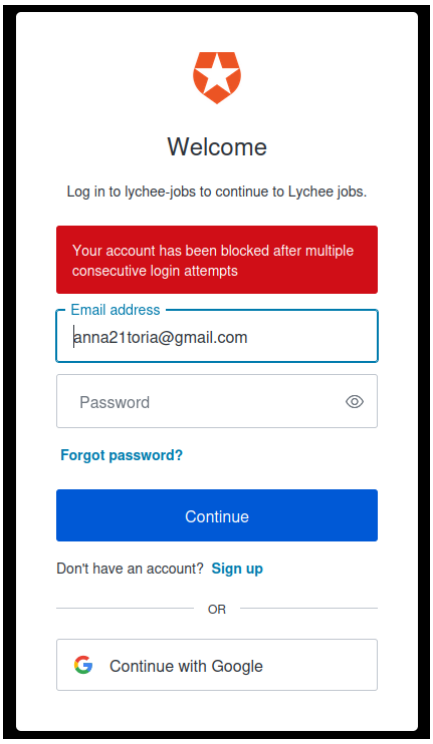
Podstawowym mechanizmem obronnym wykorzystywanym przez platformę jest mechanizm blokowania kont, który jest jednym z oczywistych i podstawowych mechanizmów stosowanych w obronie przez tymi atakami. Domyślnie ochrona jest włączona z konfiguracją, która została przedstawiona na Rys. 17. Standardowy próg wynosi 10 nieudanych prób logowania po których następuje zablokowanie dostępu do konta z adresu IP z którego pochodziły nieudane próby a właściciel konta otrzymuję wiadomość e-mail z linkiem do odblokowania konta dla danego adresu IP.



Rys. 17. Domyślna konfiguracja platformy Auth0 w ochronie przed atakami siłowymi
Źródło: opracowanie własne

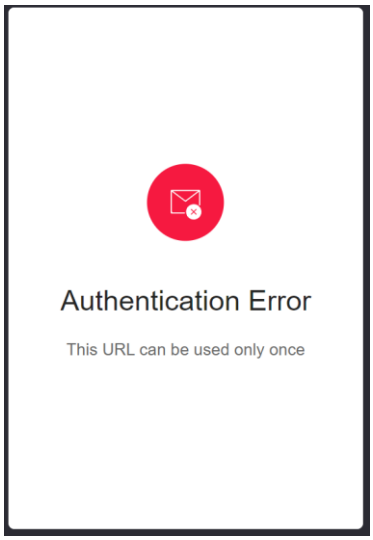
A zatem została podjęta próba dziesięciokrotnego zalogowania się błędnym hasłem, która zakończyła się zablokowanie konta. Dalsze próby zalogowania się do zablokowanego konta zakończone były informacją o tym, że dane konto jest aktualnie zablokowane (Rys. 18). Warto zwrócić uwagę na fakt, że informacja ta pojawia się niezależenie od tego czy podane hasło jest prawidłowe czy nie. W przeciwnym przypadku atakujący mógłby odczekać do momentu odblokowania konta i uzyskać

dostęp, jeśli jego właściciel nie zmieniłby hasła. Jest to zachowanie, które z jednej strony daje właścicielowi konta informację, dlaczego nie mógł się zalogować mimo wprowadzenia poprawnego hasła, jednak atakującemu może dać tylko informację o adresie e-mail, dla którego istnieje konto – jeśli takiej informacji nie posiada.



Rys. 18. Informacja o zablokowanym koncie po włączeniu mechanizmu blokady
Źródło: opracowanie własne

Po odczekaniu kilku minut, dalsza próba zalogowania się nie jest możliwa, ponieważ konto zostało zablokowane na 30 dni. Istnieje jednak możliwość wcześniejszego odblokowania przez administratora poprzez manualne odblokowanie lub podniesienie progu, zmianę hasła przez właściciela lub poprzez link, który otrzymał w wiadomości e-mail. Otrzymany link w wiadomości e-mail spełnia wymagania bezpieczeństwa, ponieważ jest unikalny a dodatkowo może on być użyty tylko raz (Rys. 19).



Rys. 19. Ponowna próba skorzystania z linku do odblokowania konta
Źródło: opracowanie własne

Przedstawiona na Rys. 20 historia odpowiedzi z każdej nie udanej próby logowania zawiera informacje o czasie RTT, który przedstawia, ile czasu zajmuję dostarczenie odpowiedzi od serwera do klienta. Jak widać czasy te różnią się między sobą, ale nie obserwujemy żadnego przyrostu z każdą kolejną próbą logowania. Wobec tego można wywnioskować, że platforma nie stosuję opóźnienia przyrostowego po każdej nie udanej próbie logowania, które jeszcze bardziej skutecznie mogłoby zapobiec próbą ataków siłowych wykonywanych tylko przez jeden wątek i zapobiegać luką bezpieczeństwa, które związane są z blokowaniem kont.

Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body
10/29/23, 11:42:58 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	492 ms	20,358 bytes
10/29/23, 11:43:03 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	321 ms	20,358 bytes
10/29/23, 11:43:06 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	386 ms	20,358 bytes
10/29/23, 11:43:10 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	269 ms	20,358 bytes
10/29/23, 11:43:14 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	368 ms	20,358 bytes
10/29/23, 11:43:18 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	404 ms	20,358 bytes
10/29/23, 11:43:22 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	343 ms	20,358 bytes
10/29/23, 11:43:29 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	327 ms	20,358 bytes
10/29/23, 11:43:58 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	406 ms	20,358 bytes
10/29/23, 11:44:35 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	281 ms	20,358 bytes
10/29/23, 11:44:45 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	483 ms	19,725 bytes
10/29/23, 11:59:14 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	416 ms	19,725 bytes
10/29/23, 11:59:36 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	284 ms	19,725 bytes
10/29/23, 11:59:42 PM	POST	https://lychee-jobs.eu.auth0.com/u/login?state=hK...	400	Bad Request	190 ms	19,725 bytes

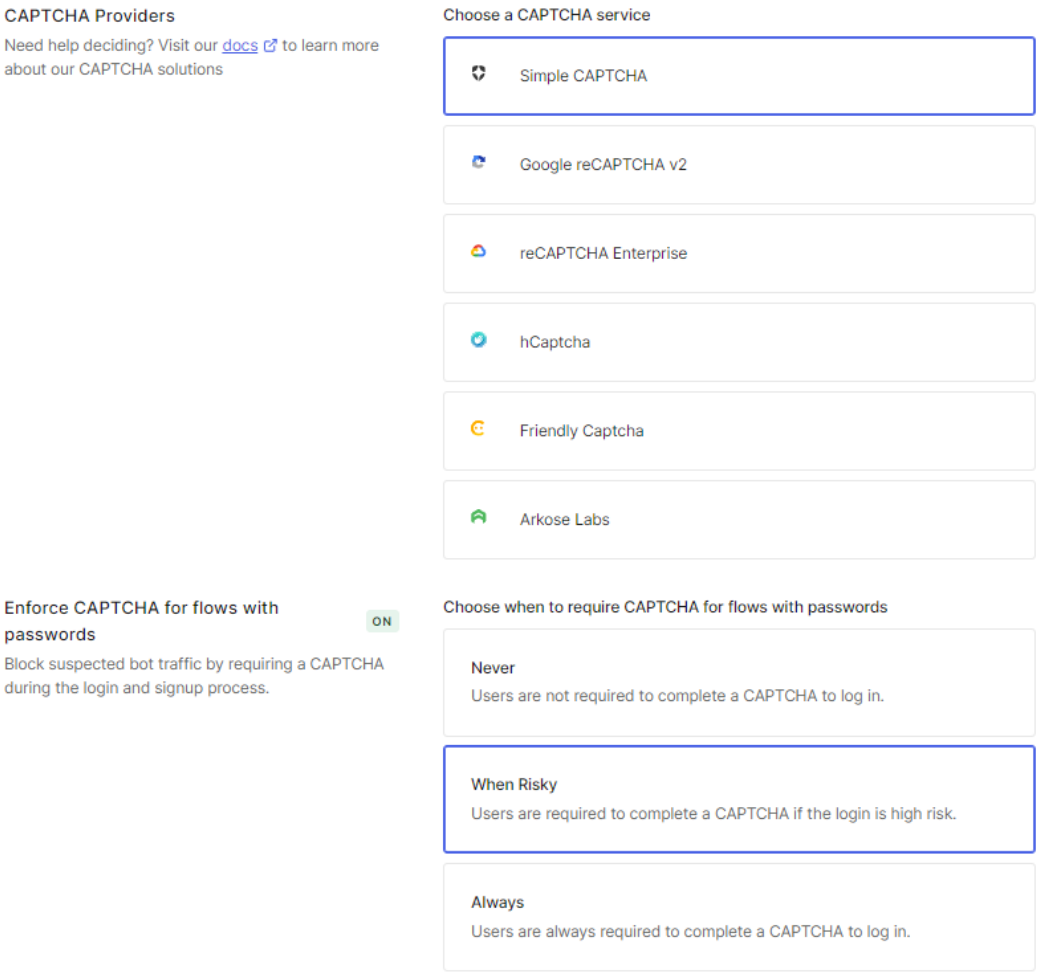
Rys. 20. Historia odpowiedzi po każdej nieudanej próbie logowania
Źródło: opracowanie własne

Adres IP z którego został osiągnięty limit nieudanych prób logowania zostaje zablokowany, ale tylko jeśli chodzi o dostęp do konta, dla którego podjęte był próby logowania. Dalej z danego adresu można zalogować się na inne konto lub tworzyć nowe konta. Aby dany adres IP został całkowicie zablokowany muszą zostać spełnione

także inne warunki takie jak liczba nieudanych prób logowania przy określonej przepustowości lub liczba rejestracji nowych kont przy określonej przepustowości, niezależnie od powodzenia tej operacji. Domyślna konfiguracja została zaprezentowana na Rys. 17. Jednak warto mieć na uwadze, że zablokowanie adresu IP może doprowadzić nawet do zablokowania dostępu całej grupie użytkowników w tym także administratora dzierżawy.

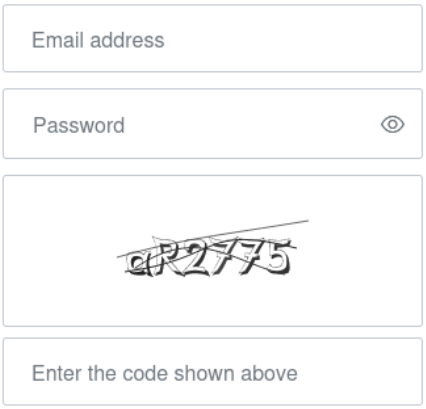
Taka konfiguracja sprawia, że konto jest zablokowane tylko dla podejrzanego adresu IP a legalny właściciel konta może bez problemu mieć dostęp (jeśli adres IP atakującego i użytkownika nie jest taki sam). Takie rozwiązanie z jeden strony chroni przed przejęciem konta a z drugiej strony nie wpływa źle na wygodę jego prawdziwego właściciela. Jednak taka konfiguracja pozwala na to, aby atak był dalej kontynuowany, ale z wykorzystaniem innego adresu IP. Istnieją różne uzasadnione przypadki, w których twórcą aplikacji będzie zależało na całkowitym zablokowaniu konta. Aby zablokować całkowicie dostęp do konta nie zależenie od adresu IP należy w tym celu należało włączyć dodatkową opcję „Account Lockout" w konfiguracji. W ten sposób platforma dostarcza możliwość dostosowania poziomu bezpieczeństwa do wygody użytkowników.

Jednak ze względu na istnienie wielu różnych scenariuszy stosowanych przez atakujących podczas siłowych ataków, mechanizm blokowania może okazać się nie skuteczny a nawet doprowadzić do kolejnej luki w bezpieczeństwie jakim jest odmowa usługi. Kolejnym rozwiązaniem oferowanym przez platformę, które może utrudniać atakującym wykonanie ataków oraz minimalizować włączenie mechanizmu blokowania kont jest mechanizm CAPTCHA. W konfiguracji przedstawionej na Rys. 21 mamy możliwość określenia, kiedy mechanizm powinien zostać uruchomiony (czy przy każdej próbie logowania czy tylko wtedy, gdy próba logowania jest podejrzana). Dodatkowo także można wybrać dostawcę rozwiązania CAPTCHA.



Rys. 21. Konfiguracja CAPTCHA w Auth0
Źródło: opracowanie własne

Przy wyborze obowiązkowego wypełnienia CAPTCHA przez każdego użytkownika w formularzu logowania, utworzenia nowego konta jak przypomnienia hasała, pojawia się dodatkowe pole do wypełnienia (Rys. 22). Dla „Simple CAPTCHA" należy wprowadzić ciąg znaków, który jest przedstawiony na obrazku, które dla utrudnienia są zniekształcone.

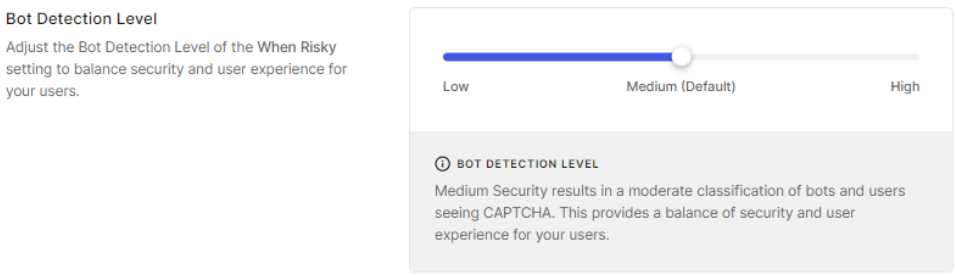


Rys. 22. Formularz logowania z obowiązkowym polem CAPTCHA
Źródło: opracowanie własne

Badając DOM, a konkretniej element odpowiedzialny za wyświetlanie obrazka można zauważyć, że znacznik IMG nie zawiera żadnych informacji, które mogłyby zdradzić ciąg znaków a dodatkowo obrazek jest w formacie base64 (Rys. 23).

Rys. 23. Struktura DOM przedstawiająca element z obrazkiem CAPTCHA
Źródło: opracowanie własne

Jednak ciągłe wymaganie uzupełnienia CAPTACH może być uciążliwe dla użytkowników lub po prostu trudne. A zatem z konfiguracji możemy także wybrać opcję „When Risky", która wyświetli CAPTACH przy wykryciu podejrzanego zachowania a nie za każdym razem (Rys. 24). Dodatkowo można tutaj także wskazać poziom wykrywania botów. Nisk poziom zapewnia, że pojawienia się CAPTACHA będzie tylko w sytuacjach, gdy podejrzenie działania bota jest bardzo wysokie, dzięki czemu prawdziwi użytkownicy nie są zmuszeni do częstego uzupełniania CAPTCHA. Natomiast wysoki częściej będzie angażował użytkowników do wypełnienia dodatkowego pola w formularzu. Wykorzystując to ustawienie można dostosować aspekt bezpieczeństwa do wygody użytkowników.

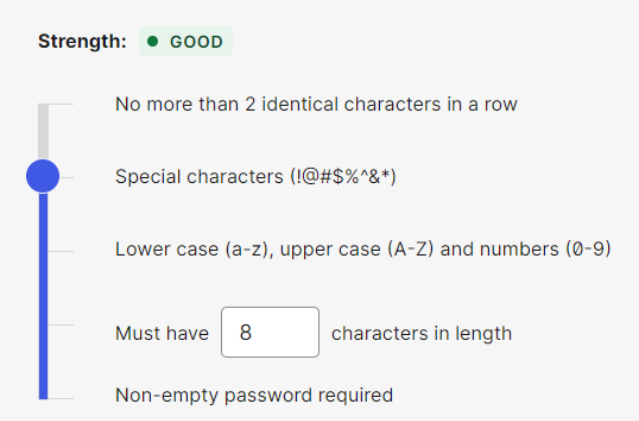


Rys. 24. Konfiguracja poziomu wykrywania botów w celu dostosowania obowiązkowego wypełniania CAPTCHA do wygody użytkowników
Źródło: opracowanie własne

Przedstawiony na Rys. 22 CAPTCH to bardzo prosta wersja, która nie wymaga dodatkowych konfiguracji jednak ma wiele wad takich jak np. brak wsparcia dla osób niewidomych. Auth0 oprócz prostego CAPTCHA oferowanego przez siebie, oferuję także skorzystanie z zewnętrznych rozwiązania, które wymagają dodatkowych działań i wprowadzenia odpowiednich danych.

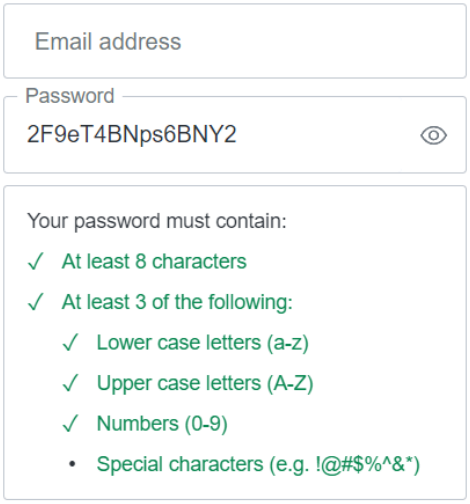
8.5 Polityka haseł

Polityka haseł jest bardzo istotnym elementem, ponieważ bardzo często użytkownicy tworzą słabe hasła, łatwe do zapamiętania dla nich a dodatkowo wykorzystują je w wielu innych aplikacjach. Dlatego skonfigurowanie dobrej polityki haseł może zapewnić, że użytkownicy będę zmuszeni do tworzenia bardziej złożonych haseł a to przełoży się na większe bezpieczeństwo. Domyślna polityka haseł stosowana przez platformę Auth0 jest ustawiona na poziomie dobrym i wymaga od użytkownika hasała o minimalnej długości 8, dużych i małych liter, znaków specjalnych i cyfr (Rys. 25). Dodatkowo można dodać kolejny poziom, który zapobiega wprowadzeniu hasała zwierającego więcej niż dwa identyczne znaki obok siebie.



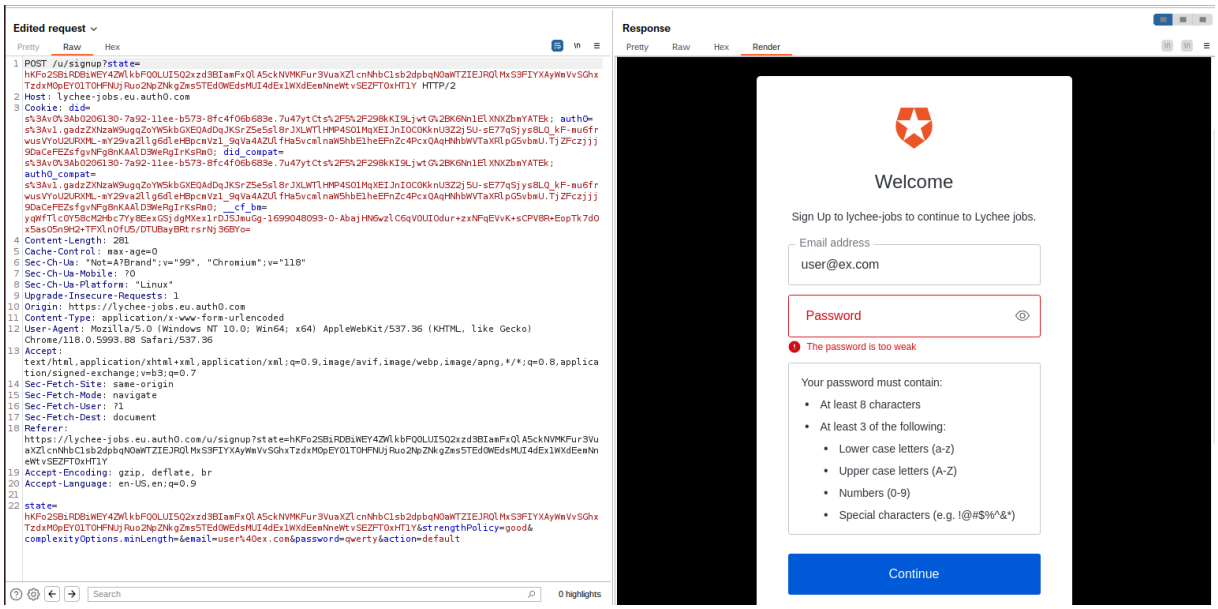
Rys. 25. Domyślna konfiguracja polityki haseł w Auth0
Źródło: opracowanie własne

Przy tak skonfigurowanej polityce, użytkownik podczas wprowadzania hasła jest informowany o stopniu spełnienia warunków (Rys. 26) oraz nie może utworzyć konta bez ich spełnienia. Wymaganie co do hasła są sprawdzane po stronie klienta i bez ich spełnienia żądanie nie jest wysyłane do serwera.



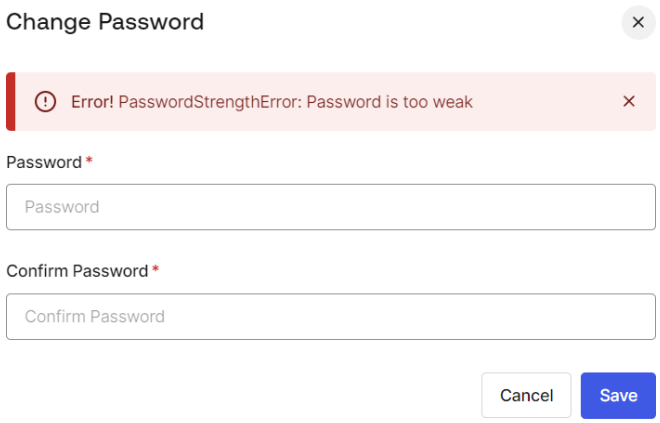
Rys. 26. Informowanie użytkownika o stopniu spełniania polityki haseł podczas tworzenia nowego konta Źródło: opracowanie własne

Ważnym aspektem jest także sprawdzenie czy hasła są także sprawdzane po stronie serwera. W tym celu, zostało przechwycone żądanie utworzenia konta z hasałem spełniającym wymagania i zmiennie go na słabe hasło. Jak można zauważyć na Rys. 27, hasło jest także poddane walidacji po stronie serwera a użytkownik dostaję informację o wprowadzeniu słabego hasła.



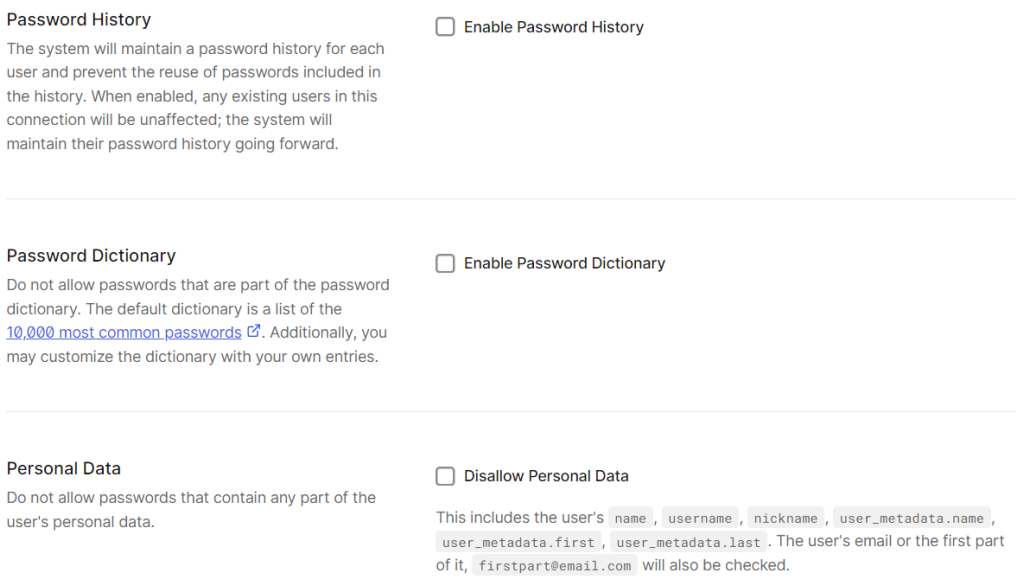
Rys. 27. Próba utworzenia nowego konta z hasłem nie spełniającym ustawionej polityki haseł
Źródło: opracowanie własne

Administrator także nie ma możliwość utworzenia użytkownikowi słabego hasła z poziomu panelu platformy Auth0 (Rys. 28).



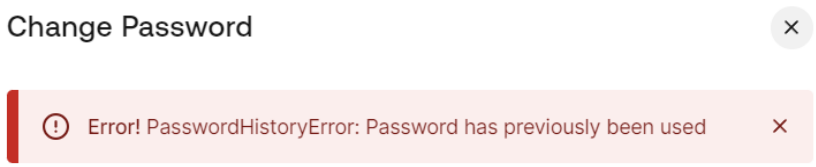
Rys. 28. Próba utworzenia słabego hasała z poziomu panelu administratora dzierżawy
Źródło: opracowanie własne

Jednak mimo spełnieniu wszystkich wymagań użytkownicy mogą tworzyć hasała, które będą podatne na ataki. Wobec tego, platforma także udostępnia dodatkowe zabezpieczenia, które będę nakładać na użytkownika utworzenia bardziej kreatywnych i skomplikowanych haseł (Rys. 29).

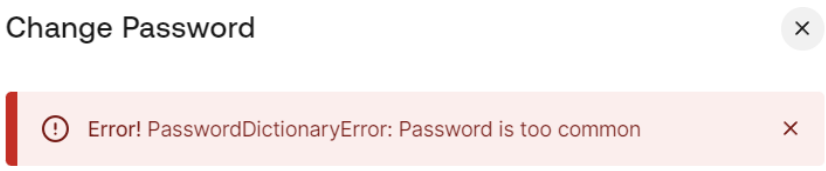


Rys. 29. Dodatkowe mechanizmy nakładające na użytkowników dodatkowe zasady tworzenia haseł
Źródło: opracowanie własne

Należą do nich m. in. prowadzenie historii haseł, które zapobiega przed użyciem hasła, które było już wykorzystane przez danego użytkownika (Rys. 30). Wymusza to na użytkowników przy zmianie hasła stworzenia całkiem nowego ciągu a nie powielania haseł, które bardzo prawdopodobnie używa także w innych miejscach. Uniemożliwienie wykorzystania haseł, które znajdują się w słowniku popularnych haseł z możliwość jego rozszerzenia o własny słownik (Rys. 31) oraz także zablokowanie możliwości utworzenia hasła zawierającego dane użytkownika (Rys. 32).

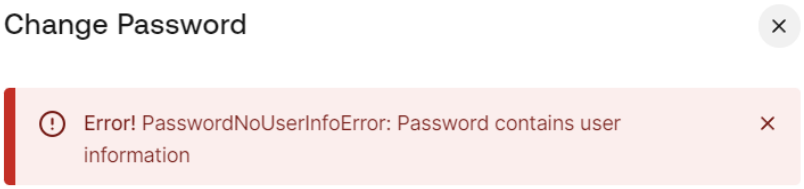


Rys. 30. Niepowodzenie podczas próby zmiany hasła na hasło, które jest zapamiętane w historii danego użytkownika
Źródło: opracowanie własne



Rys. 31. Niepowodzenie podczas próby zmiany hasła na hasło znajdujące się w słoniku haseł podatnych na złamania

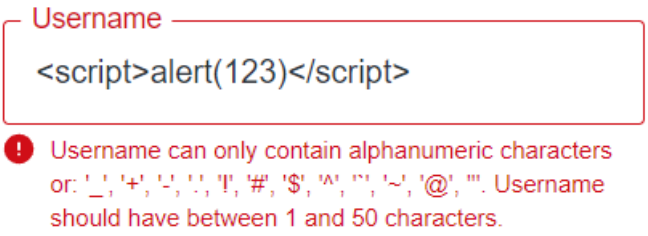
Źródło: opracowanie własne



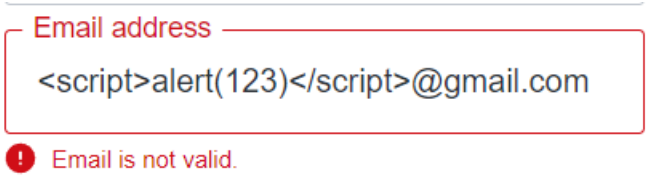
Rys. 32. Niepowodzenie podczas próby utworzenia hasła, w którym znajdują się dane użytkownika
Źródło: opracowanie własne

8.6 Zabezpieczenia przed atakami XSS i wstrzykiwaniem SQL

Aplikacje webowe, które wykorzystują dane wprowadzane przez innych użytkowników mogą być narażone na taki typu XSS. Jeśli nie zostaną wprowadzone odpowiednie mechanizmy sprawdzające dane pochodzące z zewnątrz, użytkownicy odwiedzający potencjalnie bezpieczną aplikację mogą być narażeni na wyświetlenie danych, które zostały zainfekowane niebezpiecznym skryptem. Zabezpieczenie przed tego typem atakiem jest bardzo istotne dla platformy, ponieważ możliwość wstrzyknięcia skryptu poprzez pole przeznaczone do wprowadzenia adresu e-mail lub nazwy użytkownika może zainfekować panel administratora lub inny system, w którym są wykorzystywane te dane. W związku z tym, z poziomu formularzy zostało sprawdzone czy można przesłać do serwera zainfekowane dane i czy dane żądanie zostanie zrealizowane pomyślnie.

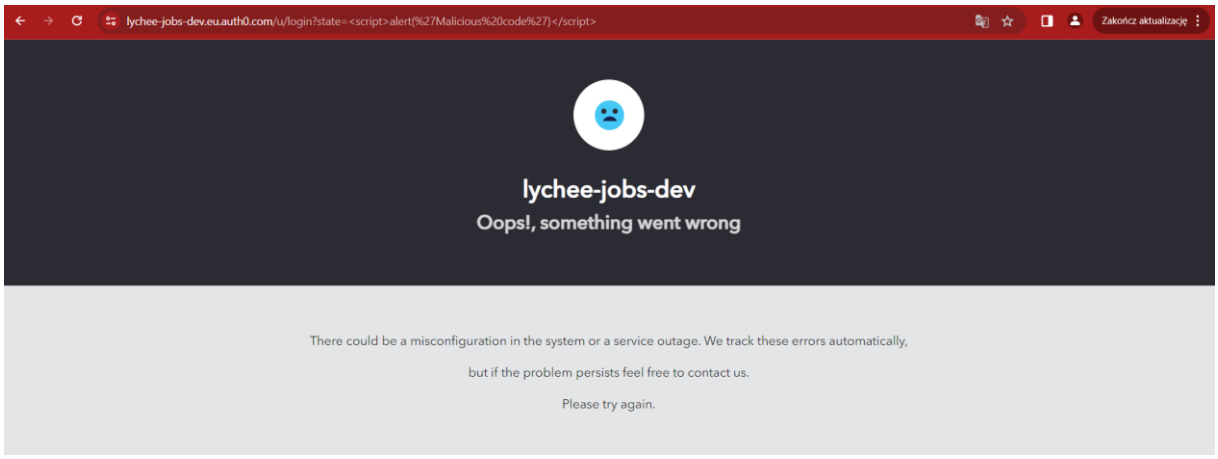


Rys. 33. Próba wstrzyknięcia skryptu do kontrolki z nazwą użytkownika
Źródło: opracowanie własne



Rys. 34. Próba wstrzyknięcia skryptu do kontrolki z adresem e-mail użytkownika
Źródło: opracowanie własne

Jak widać na Rys. 33 i Rys. 34 wprowadzone dane nie przeszły pomyślnie walidacji, ponieważ zawierały niedozwolone znaki, takie które mogą zostać wykorzystane do ataku XSS. Dodatkowo także przechwycone i zmodyfikowane żądanie nie zostało zrealizowane pomyślnie co oznacza, że także walidacja danych jest wykonywana po stronie serwera. Obserwując komunikaty o błędach można zauważyć istotną informację o dozwolonych znakach. Także i przeprowadzenie odbitego ataku XSS nie jest możliwe (Rys. 35).



Rys. 35. Próba wykonania odbitego ataku XSS
Źródło: opracowanie własne

Testowanie mające na celu ocenić rozwiązanie pod względem możliwość przeprowadzenia ataków wstrzykiwania SQL zostało przeprowadzone manualnie bez automatyzacji ze względu na to, że przeprowadzanie aktywnych testów penetracyjnych na usługach platformy Auth0 może być wykonane tylko po uzyskaniu zgody. Jednak z samej informacji o błędzie na Rys. 33 wynika, że wszelkie znaki, które by umożliwiały wstrzyknięcie SQL są nie dozwolone a dodatkowo wprowadzenie danych w innym formacie niż adres e-mail także skutkuje zwróceniem błędu. Nie jest oczywiście wiadome czy po stronie serwera są wdrożone odpowiednie mechanizmy zapobiegające temu atakowi oraz czy kod z zapytaniami SQL jest pisany według dobrych praktyk, ale sam mechanizm walidacji po stronie klienta i serwera jest już bardzo dobrą warstwą zabezpieczającą.

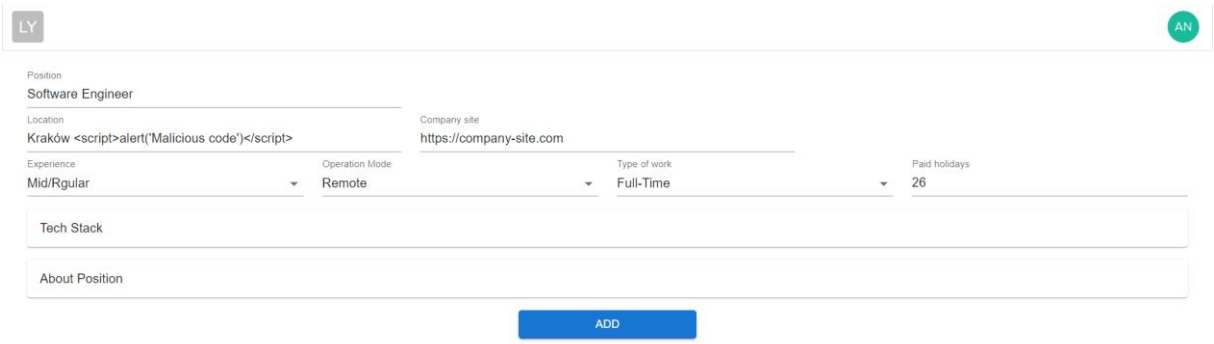
9. Analiza bezpieczeństwa React.js i Next.js

9.1 Zbadanie podatności na ataki typu XSS

W celu przeprowadzenia ataku XSS, atakujący będzie poszukiwał wszelkich luk tam, gdzie aplikacja zezwala na wprowadzenie danych z zewnątrz a następnie wykorzystuję je do generowania DOM. Analiza skupia się na wszelkich możliwych sposobach generowania DOM przy wykorzystaniu informacji wprowadzanych przez użytkowników, czyli danych, które są potencjalnie traktowana jako niebezpieczne i sprawdzeniu ich pod kątem możliwość wprowadzenia informacji, które mogą przyczynić się do niepożądanego działania aplikacji.

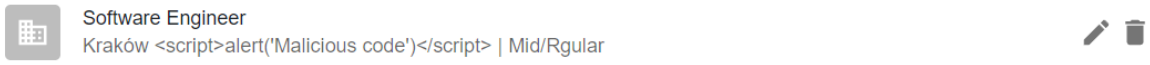
9.1.1 Osadzanie treści w składni JSX

Jednym z najprostszych i zalecanych sposób wyświetlania dynamicznych danych w DOM w rozwiązaniach opierających się o bibliotekę React jest wykorzystanie składni JXS, która pozwala na wygodne pisanie kodu HTML w JavaScript. W tym celu dane, które mają zostać wyświetlone są wprowadzane w nawiasach klamrowych.



Rys. 36. Przykład wprowadzenia złośliwego kodu z zewnątrz przez atakującego
Źródło: opracowanie własne

W przedstawionym na Rys. 36 scenariuszu ataku, atakujący próbuję poprzez udostępniony mu formularz wprowadzić niebezpieczny skrypt, który nie jest częścią legalnego kodu aplikacji. Jeśli atak zostałby przeprowadzony pomyślnie, zainfekowana część danych wprowadzona przez atakującego powinna zostać potraktowana jako skrypt a nie jako ciąg znaków. Jak można zauważyć na Rys. 37, zainfekowane dane zostały potraktowane jako ciąg znakowy co oznacza, że atak nie został przeprowadzony pomyślnie.

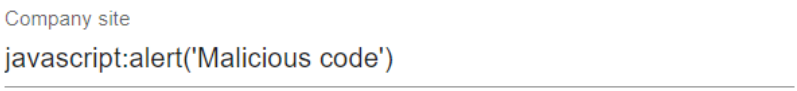


Rys. 37. Wyświetlenie zainfekowanych danych w przeglądarce ofiary
Źródło: opracowanie własne

Takie zachowanie jest spowodowane tym, że biblioteka React przed wygenerowaniem konwertuje wszystkie dane wraz z niebezpiecznymi znakami, które znajdują się w składni JSX w nawiasach klamrowych na zwykły tekst. Wobec tego, generowanie dynamicznych danych w ten sposób jest bardzo bezpieczne i co ważne, jeśli tylko to możliwe każdy kto rozwijaj aplikację webową w technologii React, powinien w ten sposób umieszczać niezaufane dane w DOM. Jest to bardzo ważne wsparcie bezpieczeństwa ze strony biblioteki, które dodatkowo dba o to, aby zapobiegać atakom XSS, jeśli oczyszczenie danych zewnętrznych zostałoby pominięte podczas tworzenia aplikacji. React także udostępnia mechanizm uniemożliwiający wstrzyknięcie szkodliwych stylów.

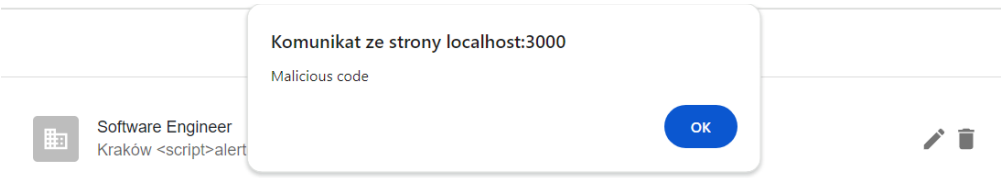
9.1.2 Niebezpieczne schematy URI

Poprzedni przykład w wyżej przedstawionym podrozdziale pokazywał jak w bezpieczny sposób można osadzić niezaufane dane w DOM i wyświetlić je jako zwykły tekst. Jednak zdarzają się takie sytuacje, kiedy dane zewnętrzne muszą zostać osadzone nie w znaczniku HTML jako tekst, który jest widoczny w przeglądarce, ale jako adres URL, który ma prowadzić użytkownika w inne miejsce w Internecie.



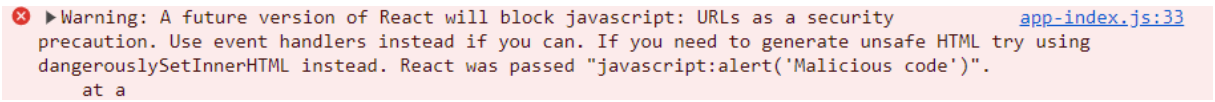
Rys. 38. Wprowadzenie zainfekowanych danych przez atakującego zamiast poprawnego adresu URL
Źródło: opracowanie własne

Przedstawiony na Rys. 38 scenariusz ataku, zakłada, że użytkownik podczas próby przejścia pod wskazany adres URL, wykona nieświadomie złośliwy kod, który atakujący podał w formularzu zamiast bezpiecznego adresu URL.



Rys. 39. Wykonanie złośliwego kodu wstrzykniętego do atrybutu „href" znacznika kotwicy
Źródło: opracowanie własne

Jak widać na Rys. 39, atakującemu udało się wstrzyknąć zainfekowany kod do atrybutu „href" mimo, że dane atakującego zostały dodane do DOM w taki sam sposób jak w pierwszym przedstawionym scenariuszu ataku, czyli poprzez nawiasy klamrowe w składni JSX. Jak można zauważyć atak został pomyślnie przeprowadzony ze względu na wykorzystanie luki związanej z możliwością użycia schematu 'javascript:'. Jest to bardzo przestarzała funkcja, która w przyszłości zostanie zablokowana z poziomu przeglądarek, ale na dziś jest poważną luką, o której istnieniu React jest bardzo świadomy i wykrywa użycie tej funkcji poprzez ostrzeżenie w konsoli przeglądarki przedstawionym na Rys. 40. Jednak jest to tylko ostrzeżenie dla programistów a sama biblioteka React nie dostarcza rozwiązania eliminującego tą poważną lukę.



Rys. 40. Wykrycie osadzenia niebezpiecznego bloku kodu w atrybucie href przez React
Źródło: opracowanie własne

Jednym ze sposobów na całkowitą eliminację tej luki jest oczywiście niekorzystanie z adresów URL pochodzących z zewnątrz. Oczywiście takie rozwiązanie nie zawsze jest możliwe. Innym rozwiązaniem minimalizującym tą lukę jest stosowanie ograniczeń na użytkownikach poprzez danie im możliwość do wprowadzania tylko części adresu URL, jeśli funkcjonalność z góry określa do jakiej aplikacji lub witryny internetowej ma prowadzić adres. Jeśli jednak żaden z przedstawionych sposób nie jest możliwy do zastosowania wtedy należy wszelkie dane zewnętrzne odnoszące się do adresów URL potraktować z góry jako niebezpieczne i zadbać o ich sprawdzenie i oczyszczenie przed

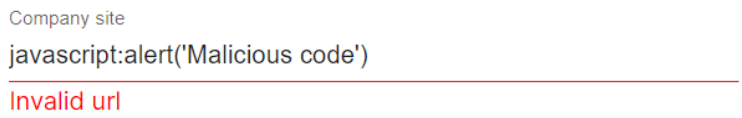
wygenerowaniem DOM. W tym celu należy napisać kod, który sprawdzi wprowadzone dane z zewnątrz i je oczyści w taki sposób, aby ograniczyć potencjalne zagrożenia. Aby taki mechanizm zaimplementować należy wyznaczyć kryteria, wobec których dane będą uznawane za bezpieczne lub nie. W tym podejściu adres URL zostałby sprawdzony pod względem użycia niebezpiecznego schematu i uznany za niebezpieczny, jeśli taki schemat zostałby wykryty. Jednak to rozwiązanie będzie tylko skuteczne dla znanych zagrożeń a to może spowodować poważną lukę, jeśli atakujący znajdą inny schemat umożliwiający przeprowadzenie ataku. Wobec tego najlepszym sposobem jest sprawdzanie adresów w kontekście spełniania dobrych zasad i tylko takie traktować jako bezpieczne. Takie podejście stosuję platforma programistyczna Angular, który w przeciwieństwie do biblioteki React.js, dostarcza mechanizm oczyszczający adresy.

```
interface SanitizedUrl {
  isSecure: boolean,
  url: string,
}
const SAFE_URL_PATTERN = /^(?:(?:https?|mailto|ftp|tel|file|sms):|[^&:/?#]*(?:[/?#]|$))/gi;
const DATA_URL_PATTERN =
/^data:(?:image\/(?:bmp|gif|jpeg|jpg|png|tiff|webp)|video\/(?:mpeg|mp4|ogg|webm)|audio\/(?:
mp3|oga|ogg|opus));base64,[a-z0-9+\/]+=*$/i;
function _sanitizeUrl(url: string): SanitizedUrl {
  if (url === "null" || url.length === 0 || url === "about:blank")
    return {isSecure: true, url: "about:blank"};
  if (url.match(SAFE_URL_PATTERN) || url.match(DATA_URL_PATTERN))
    return {isSecure: true, url: url};
  return {isSecure: false, url: `unsafe:${url}`};;
}
export function sanitizeUrl(url = "about:blank"): SanitizedUrl {
  return _sanitizeUrl(url.trim());
}
```

Listing 3. Kod oczyszczający adresu URL na podstawie mechanizmu stosowanego przez Angular
Źródło: opracowanie własne na podstawie [33]

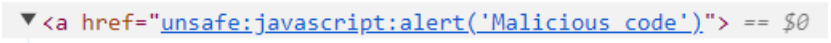
Przedstawiony kod (Listing 3) prezentuje oczyszczanie adresów URL na podstawie mechanizmu, który stosuje sam Angular. Rozwiązanie dodatkowo zostało rozszerzone o zwracanie nie tylko bezpiecznego adresu, ale także flagi informującej czy wprowadzony adres był bezpieczny czy nie. Daje to możliwość przeprowadzenia dodatkowego sprawdzenia adresów już podczas wprowadzania ich przez użytkownika. Jest to dobra praktyka, ponieważ wszelkie dane pochodzące z zewnątrz powinny być

sprawdzone na każdym etapie, aby zapobiegać wystąpieniu potencjalnych luką. Wobec tego dane powinny zostać poddane walidacji przed zatwierdzeniem jak i ponownie sprawdzone przed ich wygenerowaniem w DOM. Po modyfikacjach, wprowadzenie zainfekowanego adresu URL nie jest już możliwe ze względu na walidacje, czyli sprawdzanie danych już na etapie ich wprowadzania w formularzu i zablokowaniu możliwość wprowadzenia podejrzanych danych mogących przyczynić się do przeprowadzenia ataku (Rys. 41).



Rys. 41. Efekt działania walidacji sprawdzającej wprowadzony adres URL przez użytkownika pod względem spełnienia dobrych wzorców
Źródło: opracowanie własne

Jednak mimo braku możliwość przesłania adresu URL nie pasującego do dobrych znanych wzorców, adresy są i tak dodatkowo poddane oczyszczeniu przed wygenerowaniem DOM (Rys. 42).



Rys. 42. Efekt oczyszczenia adresu przed wygenerowaniem DOM
Źródło: opracowanie własne

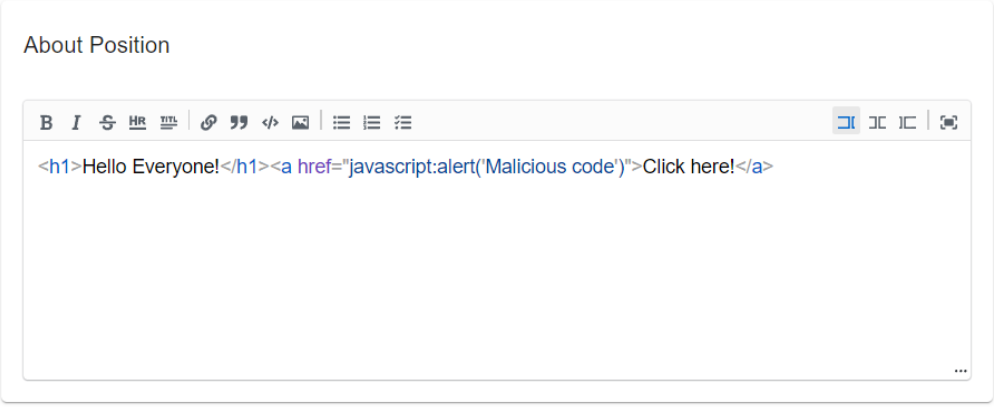
9.1.3 Generowanie niezaufanego kodu HTML w DOM

Kolejnym ważnym aspektem, na który należy także zwrócić uwagę w analizie potencjalnych zagrożeń na ataki typu XSS jest dynamiczne generowanie elementów HTML w DOM. Jest to bardzo popularne podejście, jeśli aplikacja webowa daje użytkownikom możliwość tworzenia treści z własnym formatowaniem np. wpisów do blogu. Takiego typu danych nie można osadzić w składni JSX, ponieważ zostaną one przekonwertowane na ciąg znaków co z jednej strony wpływa dobrze na bezpieczeństwo i eliminuję lukę XSS, ale z drugiej strony formatowanie treści nie zostanie zachowane i dane zostaną wyświetlone w bardzo nieczytelnej i nieprzyjemnej formie dla człowieka. Biblioteka React pozwala na wygenerowanie takich treści z zachowaniem formatowania poprzez udostępnienie atrybutu „dangerouslySetInnerHTML” (Listing 4), ale jak sama nazwa wskazuje nie jest to bezpieczne rozwiązanie i naraża aplikację na ataki XSS.

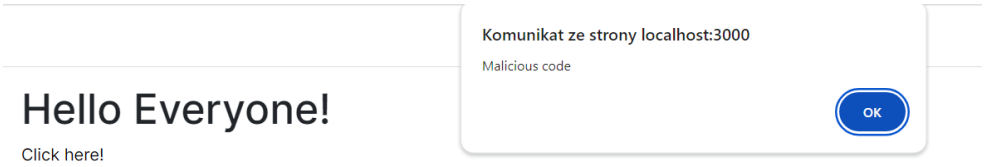
```
return(
      <>
      {offer &&
           <div className="container">
               <div className="row">
                   <div className="col-12" dangerouslySetInnerHTML={{__html:
offer.description}}></div>
               </div>
           </div>}
      </>
   )
```

Listing 4. Niebezpieczny sposób generowanie dynamicznych treści z zachowaniem formatowania
Źródło: opracowanie własne

Jeśli w ten sposób zezwoli się na ładowanie danych z zewnątrz, atakujący może bez większego problemu przekazać niebezpieczny kod (Rys. 43), ponieważ wszelkie znaczniki HTML zostaną wygenerowane poprawianie w DOM (Rys. 44).



Rys. 43. Wprowadzenie zainfekowanego kodu w edytorze składni języka Mrakdown
Źródło: opracowanie własne



Rys. 44. Wygenerowanie strony z zachowaniem formatowania użytkownika i zainfekowanym kodem
Źródło: opracowanie własne

Dlatego i tutaj wszelkie dane powinny zostać oczyszczone i przed wysłaniem ich na serwer jak i przed wygenerowaniem ich w DOM. Warto tutaj zwrócić uwagę, że często programiści korzystają z gotowych rozwiązań takich jak edytory do treści Markdown,

które w swoich dokumentacjach piszą o posiadaniu funkcji do oczyszczania, ale nie należy na nich polegać w tej kwestii dlatego, że często nie są one dopracowane. Jednym ze sposobów na oczyszczanie danych i zminimalizowanie podatności na atak XSS jest skorzystanie z biblioteki dompurify. Po wprowadzeniu modyfikacji polegającym na wcześniejszym oczyszczeniu danych przy pomocy rozwiązania oferującego poprzez wcześniej wspomnianą bibliotekę (Listing 5), treść została wygenerowana poprawnie, ale już bez potencjalnie niebezpiecznego kodu.

```
const sanitizeMD = useMemo(() => {
        return offer ? DOMPurify.sanitize(offer.description) : '';
    }, [offer])

    return(
       <>
       {offer &&
            <div className="container">
                <div className="row">
                    <div className="col-12" dangerouslySetInnerHTML={{__html:
sanitizeMD}}></div>
                </div>
            </div>}
       </>
    )
```

Listing 5. Przykład przeprowadzenia procesu oczyszczania danych z wykorzystanie biblioteki dompurify
Źródło: opracowanie własne

```
▼<div class="container">
  ▼<div class="row">  flex
    ▼<div class="col-12">
        <h1>Hello Everyone!</h1> == $0
        <a>Click here!</a>
      </div>
    </div>
  </div>
```

Rys. 45. Efekt oczyszczenia danych z wykorzystaniem biblioteki dompurify
Źródło: opracowanie własne

Biblioteka dompurify jest jednym z dobrych przykładów istniejących rozwiązań, które pomagają w minimalizowaniu wystąpień luk umożliwiających przeprowadzenia ataków XSS, jednak nie gwarantują niezawodności. Wobec tego, należy ciągle testować

narażone części aplikacji a także dbać o to, aby aplikacja korzystała z najnowszej wersji biblioteki używanej do oczyszczenia danych niezaufanych. Dodatkowo warto korzystać z dobrych i zaufanych bibliotek niż pisać własne funkcję służące do oczyszczania danych, gdyż napisanie dobrego rozwiązania wymaga dużej i specjalistycznej wiedzy oraz ciągłego udoskonalania w zapewnieniu ochrony przed atakującymi, którzy cały czas szukają nowych luk.

Biblioteka React także udostępnia możliwość odwołania się bezpośrednio do elementu DOM poprzez referencję korzystając z „useRef". W ten sposób można modyfikować zawartość elementu w sposób imperatywny np. wykorzystując w tym celu funkcję „innerHTML" czy „append", które będą interpretować poprawianie wszelkie znaczniki języka HTML. Jeśli takie modyfikację mają na celu wyświetlanie niezaufanych danych w postaci tekstowej, warto tutaj posłużyć się funkcją „innerText", która bezpiecznie wygeneruję wszystko w postaci tekstowej w przeciwnym wypadku należy zadbać o odpowiednie oczyszczenie danych lub jeśli to możliwe unikać takiego sposobu generowania DOM.

9.1.4 Wykorzystanie danych w formacie JSON

Nowoczesne aplikację często wykorzystują narzędzia do zarządzania stanem aplikacji, aby dostarczyć użytkownikowi jak najlepsze rozwiązania. Jednym z popularnych narzędzi stosowanych wraz z rozwiązaniami bazującymi na bibliotece React jest Redux [34], który nie wątpliwie daję duże możliwości, ale także może przyczynić się do powstania poważnej luki w bezpieczeństwie dla aplikacji wykorzystujących SSR, które muszą dostarczyć początkowy stan aplikacji w postaci danych w formacie JSON. Dane w takim formacie muszą zostać podane operacji serializacji, która często odbywa się poprzez skorzystanie z „JSON.stringfiy". Naraża to aplikacje na ataki typu XSS w kontekście warstwy HTML-a. Sam Redux dostarcza mały mechanizm zapobiegający bardzo prostym atakom jednak nie jest to wystarczające rozwiązanie, jeśli dane nie zostaną wcześniej oczyszczone (Listing 6). Redux jest tylko jednym z przykładów możliwość wystąpienia takiego problemu. Ogólny problem sprowadza się do tego, że jeśli taki kod trafi do struktury DOM to znaczniki znajdujące się w wszelkich danych testowych zostaną poprawianie zinterpretowane. Mimo, że na pierwszy rzut oka taki kod nie wygląda na niebezpieczny

w środowisku JavaScript jednak może okazać się poważną luką od strony HTML-a [34].

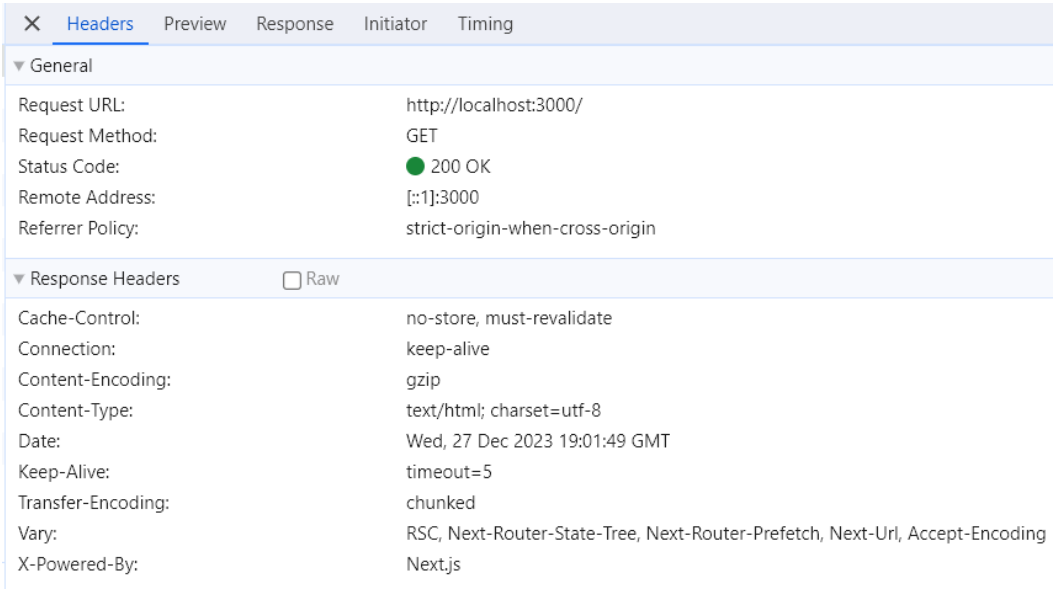
```
<script>
   window.__PRELOADED_STATE__ = ${JSON.stringify(state)
                                  .replace(/</g, "\\u003c")}
</script>
```

Listing 6. Prosty mechanizm zapobiegający niektórym problem związanym z serializacją danych w formacie JSON w warstwie HTML-a
Źródło: [34]

Inną ważną sprawą jest także wykorzystywanie mechanizmu serializacji w środowisku Node.js, które może narazić aplikację na wstrzyknięcie do kodu nielegalnych funkcji, które mogą przyczynić się do nieupoważnionych zmian lub nawet być powierzchnią ataków typu DoS. Może tak się zdarzyć, jeśli skorzystamy z narzędzi, które potrafią poprawianie serializować każdy obiekt. Aby zapobiegać tego typu podatności, należy ograniczać wykorzystanie użycia serializacji. Zadbać o to, aby dane zostały oczyszczone a wrażliwe znaki odpowiednio zakodowane. Należy także dobrać odpowiednie narzędzie do operacji na danych w formacie JSON a wybór powinien odbywać się w kontekście środowiska, w którym będzie wykorzystywane.

9.2 Nagłówki bezpieczeństwa

Nagłówki bezpieczeństwa są ważnym aspektem zapewnienia ochrony pomiędzy klientem a serwerem, które informuję przeglądarki o tym jak powinny się zachowywać w określonych przypadkach. Rys. 46 przedstawia nagłówki odpowiedzi HTTP, które są ustawiane domyślnie po wygenerowaniu szablonu aplikacji. Początkowo są one wyłączone i aplikacja jest podatna na wiele ataków takich jak XSS czy przechwytywanie kliknięcie.



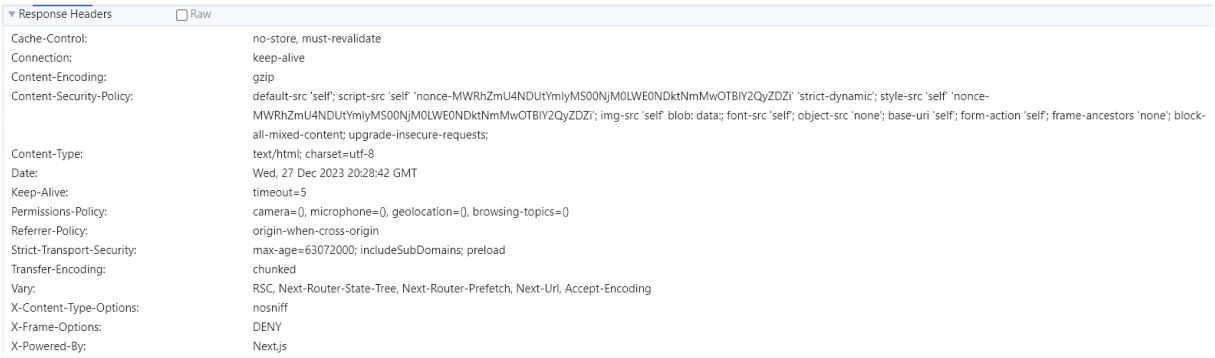
Rys. 46. Domyślne nagłówki odpowiedzi HTTP aplikacji Next.js
Źródło: opracowanie własne

Aby włączyć odpowiednie nagłówki bezpieczeństwa w aplikacji Next.js należy uwzględnić je w pliku konfiguracyjnym next.config.js, który znajduję się w głównym katalogu projektu. Przedstawiona na Listing 7 konfiguracja odnosi się do wszystkich tras w aplikacji.

```
/** @type {import('next').NextConfig} */
const nextConfig = {
    reactStrictMode: true,
    swcMinify: true,
    async headers() {
      return [
          {
            source: '/(.*)',
            headers: [
              {
                key: 'Content-Security-Policy',
                value:"default-src 'self';",
              },
            ],
          },
        ];
    },
  }
module.exports = nextConfig
```

Listing 7. Przykładowa konfiguracja nagłówków bezpieczeństwa HTTP w Next.js
Źródło: opracowanie własne

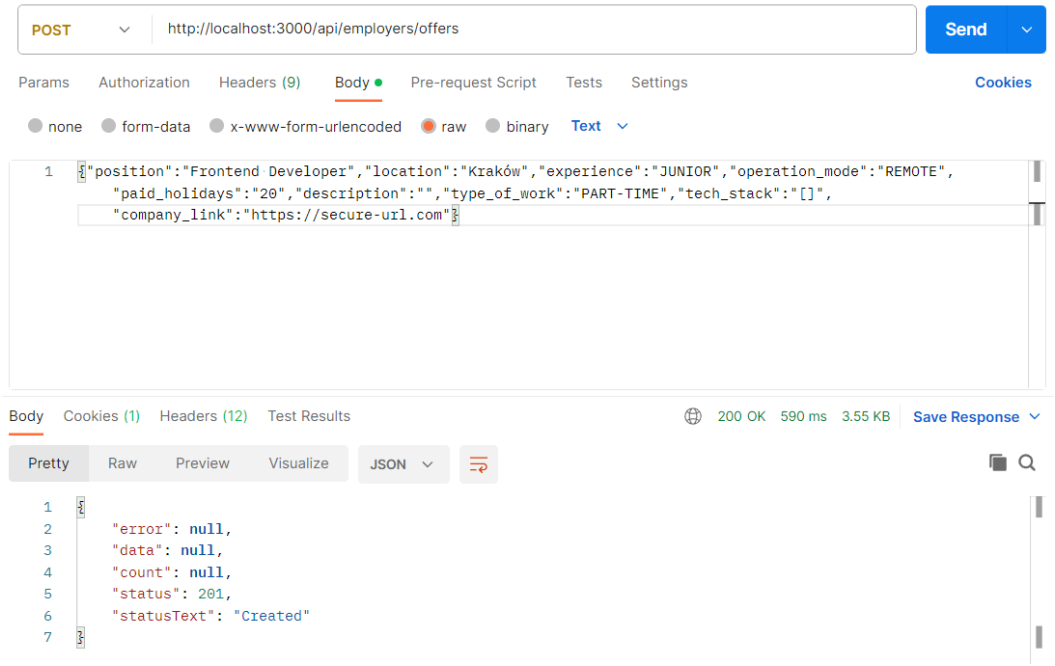
Do przedstawionej konfiguracji została dodana dyrektywa bezpieczeństwa CSP została tak zdefiniowana, że nie ma możliwości wykonanie legalnych skryptów i stylów, które są osadzone bezpośrednio w stronie. Tak rygorystyczne rozwiązanie mimo wysokiego bezpieczeństwa jest bezpośrednią przyczyną tego, że aplikacja Next.js nie będzie działać poprawianie. Aby temu zapobiec przy zachowaniu odpowiedniego poziomu bezpieczeństwa, rekomendowane jest użycie tzw. Mechanizmu wartości jednokrotnej „nonce" zamiast np. „unsafe-inline", które może narazi aplikacje na poważną lukę. W dokumentacji Next.js można znaleźć odpowiednie rozwiązanie dla wprowadzenia tego mechanizmu poprzez dodanie oprogramowania pośredniczącego. Po wprowadzeniu zalecanych modyfikacji przez dokumentacje, nagłówek odpowiedzi HTTP zawiera zdefiniowane nagłówki bezpieczeństwa wraz z mechanizmem wartości jednokrotnej, która jest ciągiem losowych znaków, które za każdym razem są inne a aplikacja działa poprawianie [35].



Rys. 47. Nagłówek odpowiedzi HTTP po zdefiniowaniu odpowiednich nagłówków bezpieczeństwa i mechanizmu „nonce"
Źródło: opracowanie własne

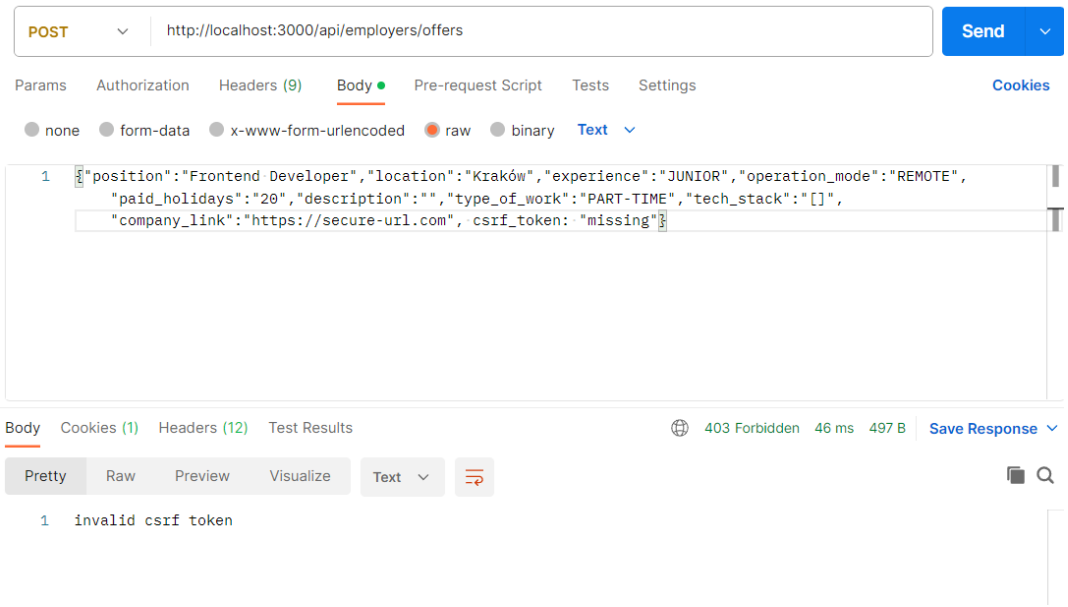
9.3 Ochrona przed atakami typu CSRF

Zaniedbanie zabezpieczeń przed atakami typu CSRF, mogą doprowadzić do wykonywania operacji w imieniu legalnego użytkownika poprzez posłużenie się jego plikiem Cookie. Na Rys. 48 został przedstawiony wynik podjętej próby wykonania operacji z wykorzystaniem pliku cookie zalogowanego użytkownika. Otrzymana odpowiedz pokazuję, że operacja zakończyła się powodzeniem.



Rys. 48. Wykonanie operacji w imieniu zalogowanego użytkownika poprzez wykorzystanie jego plików Cookie
Źródło: opracowanie własne

Jedynym z zabezpieczeń przed tego typu atakiem jest dodanie do plików cookie odpowiednich atrybutów takich jak „HttpOnly” i „SameSite”, które instruują przeglądarkę w jakich sytuacja powinna dodawać pliki cookie. Jednak dodatkowym wzmocnieniem bezpieczeństwa jest zaimplementowanie obsługi tokenów. Next.js udostępnia w tym celu dedykowane narzędzie takie jak „edge-csrf” lub „next-csrf”. Po wprowadzeniu odpowiednich zmian w kodzie aplikacji umożliwiających skorzystanie z tokenu, ponowne wysłanie żądania w imieniu innego użytkownika zakończyło się niepowodzeniem mimo posłużeniu się jego plikiem Cookie, ponieważ przesłany token nie był poprawny (Rys. 49).



Rys. 49. Próba wykonania operacji w imieniu zalogowanego użytkownika po wprowadzeniu tokenu Anti-CSRF
Źródło: opracowanie własne

9.4 Zmienne środowiskowe

Zmienne środowisko są przydatnym rozwiązanym w przechowywaniu danych, które wykorzystane są wielu miejscach w kodzie aplikacji i także zależne od docelowego środowiska, w których uruchomiana jest aplikacja m. in. deweloperskim, produkcyjnym czy testowym. Często zmienne środowiskowe służą do przechowywania danych niezbędnych do konfiguracji pewnych funkcjonalność jak np. integracja aplikacji z platformą Auth0 (Listing 1). Są to np. klucze lub tokeny, które są danymi poufnymi i nigdy nie powinny być udostępniane użytkownikowi. Istotną kwestią platformy programistycznej Next.js jest to, że jeśli nazwa zmiennej środowiskowej nie zostanie poprzedzona prefiksem „NEXT_PUBLIC” to nigdy nie zostanie udostępniona w środowisku klienta a tylko i wyłącznie będzie ona dostępna z poziomu serwera. Jest to bardzo dobre rozwiązanie, ponieważ pozwala odzieli zmienne między oboma środowiskami, ograniczając ich dostęp do konkretnego środowiska a dodatkowo tylko poprzez odpowiednie nazwanie zmiennej można ją użyć w środowisku klienta. Przez co omyłkowe nazwanie zmiennej nie spowoduje jej przekazania klientowi.

10. Wnioski

Podsumowując przeprowadzoną analizę w kontekście bezpieczeństwa jakie oferuje platforma Auth0 można stwierdzić, że na dziś jest to bardzo bezpieczne rozwiązanie z zakresu uwierzytelniania i autoryzacji. Platforma oferuje mechanizmy w celu ochrony przed różnymi atakami, które można dostosowywać w celu optymalnego zapewnienia bezpieczeństwa jak i wygody użytkowników. Po utworzeniu aplikacji w dzierżawy nie ma potrzeby wprowadzania konfiguracji, ponieważ domyślna konfiguracja jest bardzo dobrze zabezpieczona przed atakami XSS i wstrzykiwaniem SQL poprzez solidną walidację zarówno po stronie klienta jak i serwera, niestandardową nazwę tokenu Anti-CSRF i ochronę przed atakiem Clickjacking. Włączona jest także domyślnie ochrona przed siłowymi atakami. Integracja z platformą jest bardzo łatwa i nie wymaga skomplikowanych konfiguracji, które mogłyby przyczynić się do wystąpienia błędów. Udostępniony panel administratora jest bardzo intuicyjny a wszelkie ustawiania są opatrzone przejrzystym objaśnieniem. Dodatkowo panel administratora jest wyposażony w monitoring i dziennik alertów. Auth0 ma także bardzo dobrze opracowaną dokumentacje a także wiele treści dotyczących obszaru bezpieczeństwa, z których można czerpać solidną wiedzę.

To co może być niekorzystne dla twórców aplikacji jest to, że z poziomu panelu dzierżawy nie ma możliwości na wprowadzenie bardziej rozbudowanej konfiguracji np. nagłówków bezpieczeństwa HTTP. Jest to oczywiście dobre pod względem dzierżawców usługi, którzy nie mają zbyt dużej wiedzy na temat konfiguracji a dzięki temu platforma zapobiega niekorzystnym konfiguracją. Jednak bardziej zaawansowani dzierżawcy mogą być w tym momencie bardziej ograniczeni i jedynie mogą liczyć na zmiany w konfiguracjach, które wdrożą specjaliści Auth0 lub są zmuszeni do wdrożenia innych alternatywnych rozwiązań jak np. własny panel logowania lub serwer pośredniczący. To także dotyczy zmian w standardach nagłówków bezpieczeństwa HTTP, które muszą zostać uwzględnione przez platformę i wdrożone w rozwiązaniu.

Innym aspektem oczywiście w kontekście darmowej wersji Auth0 jest dostarczanie tylko mechanizmu CAPTCHA i wykrywania botów do minimalizowania luk związanych z blokowaniem adresów IP, które mogą doprowadzić w szczególnych przypadkach do odmowy usługi. Jest to kluczowa kwestia, gdy projektowane usługi muszą być zorientowane na dostarczenie takiego rozwiązania, w którym

niedopuszczalne jest doprowadzenie do ataku DoS a w przeprowadzonej analizie rozwiązania Auth0 oprócz CAPTCHA i mechanizmu wykrywania botów nie zaobserwowano np. mechanizmu opóźnienia przyrostowego. Może to być krytyczny problem dla rozwiązań, które nie mogą sobie pozwolić na odmowę dostępu. Dodatkowo użytkownik jest także narażony na atak MITM przy pierwszym załadowaniu aplikacji w przeglądarce – co oczywiście można rozwiązać poprzez dodanie domeny do listy wstępnego ładowania zarządzaną przez Google.

W kontekście podatności na ataki XSS, biblioteka React dostarcza bardzo pomocny mechanizm konwertujący dane umieszczone w składni JSX na bezpieczny ciąg znaków uniemożliwiający przeprowadzenie takiego ataku i także zapobiegający wstrzykiwaniu nielegalnych stylów. Istnieją jednak takie przypadki, w których React nie udostępnia żadnych gotowych rozwiązań m. in. przy umieszczenia w DOM adresów URL i treści, które nie mają być traktowane jako tekst. Jedynym wsparciem ze strony biblioteki jest alert ostrzegawczy i nazewnictwo sugerujące zagrożenie z użycia danego rozwiązania. Tutaj niestety zagwarantowanie bezpieczeństwa jest w gestii programisty poprzez m. in. walidacje danych lub zastosowanie bibliotek do oczyszczania danych. Innym nieszablonowym sposobem na eliminacje takich zagrożeń jest przeanalizowanie innych bibliotek i platform programistycznych pod kątem istnienia takich mechanizmów i zaimplementowanie ochrony na podstawie tych rozwiązań np. Angular implementuje oczyszczanie adresów.

Innym aspektem wpływającym na bezpieczeństwo jest to, że szablon aplikacji Next.js nie dostarcza w początkowej konfiguracji żadnych zdefiniowanych nagłówków bezpieczeństwa HTTP. To zobowiązuje programistę do samodzielnego konfigurowania nagłówków, które bez znajomości dokumentacji, rekomendacji i narzędzia może przyczynić się do luki XSS. Ogólnie jest to spowodowane możliwością ustawienia bardzo rygorystycznej polityki bezpieczeństwa treści, która powoduje blokadę wbudowanych skryptów a co za tym idzie wpływa na niepoprawne działanie aplikacji. W tym momencie bardzo istotne jest wdrożenie mechanizmu wartości jednokrotnej, który jest rekomendowany w dokumentacji wraz z bardzo intuicyjnym rozwiązaniem, jednak bez rzetelnego podejścia do problemu programista może po prostu złagodzić politykę bezpieczeństwa a w konsekwencji narazić użytkowników i aplikacje na poważne zagrożenia.

Dodatkowo Next.js rozszerza to co oferuje biblioteka React m. in. o generowanie stron po stronie serwera, które pozwala ograniczyć ilość udostępnianych danych o niezbędne a także redukuje potrzebę instalacji zewnętrznych zależności np. poprzez wbudowaną obsługę tras. Jednak samo SSR oprócz zalet może także przyczynić się do luk związanych z konwertowaniem danych w formacie JSON podczas dostarczani początkowej konfiguracji rozwiązaniu do zarządzania stanem aplikacji, jeśli niezostanie zaimplementowanie oczyszczanie w odpowiedniej warstwie aplikacji.

Nie wątpliwie React i Next.js są projektowane z uwzględnię aspektów bezpieczeństwa dostarczając mechanizmy i wskazówki dla programistów a także rekomendacje w dokumentacji. Jednak to czy aplikacja webowa będzie bezpieczna w dużej mierze jest uwarunkowane tym jaką świadomość w obszarze zagrożeń ma programista i na ile zna obie technologie uwzględniając fakt, że mimo że konfiguracja Next.js jest bardzo łatwa to jest on znacznie trudniejszy w nauce niż React.

Ogólną zaletą Next.js, React.js oraz Auth0 jest to, że są to rozwiązania popularne, aktywnie rozwijane i udoskonalane przez specjalistów. Dzięki temu ewentualne wykrycie luki w tych technologiach spotka się z natychmiastową reakcją ze strony utrzymujących te technologie w celu eliminacji podatności.

Podsumowanie

Celem niniejszej pracy było wykonanie analizy pod względem bezpieczeństwa jakie oferuje Next.js oraz platforma Auth0, które mogą zostać wykorzystane do tworzenia nowoczesnych aplikacji webowych. Testy obejmowały m. in. sprawdzenie jakie możliwość i mechanizmy oferuje Next.js i React.js, które mogą zostać użyte do minimalizacji powierzchni ataków po stronie klienta takich jak XSS czy CSRF oraz czy Auth0 w wersji darmowej dostarcza bezpieczne rozwiązanie z zakresu uwierzytelniania.

Rozdziały teoretyczne miały na celu przybliżyć tematykę, z którą związana jest niniejsza praca w lepszym rozumieniu części praktycznej m. in. w kontekście bezpieczeństwa, nowoczesnych aplikacji webowych jak i zagrożeń oraz metod ich minimalizacji.

Przeprowadzana analiza oraz zaprezentowane wnioski, pokazują, że Next.js bazujący na bibliotece React.js wraz z platformom Auth0 mogą zostać wykorzystane do stworzenia bezpiecznej aplikacji webowej. Warto tutaj zaznaczyć, że oczywiście musi temu towarzyszyć odpowiednia wiedza z zakresu bezpieczeństwa jak i dobre zrozumienie wykorzystywanych technologii i ich możliwości.

Pomimo iż cel pracy został osiągnięty, jednak przeprowadzona analiza bezpieczeństwa nie gwarantuje, że za jakiś czas te rozwiązania będą nadal tak samo bezpieczne, ponieważ metody ataków ciągle zostają ulepszane i stają się coraz bardziej podstępne. Dlatego decydując się na rozwój aplikacji webowych przy wykorzystaniu tych technologii należy cały czas monitorować i oceniać ich bezpieczeństwo. W przyszłości także należałoby poddać analizie bezpieczeństwa platformę Auth0 pod względem opcji płatnych a także skupić się bardziej na aspektach bezpiecznego serwera w Next.js oraz ewentualnej integracji z platformą udostępniającą rozwiązania z zakresu baz danych.

Bibliografia

[1] Bendovschi A., Cyber-Attacks – Trends, Patterns and Security Countermeasures, Oxford, Wdham College, United Kingdom, 2015

[2] Hoffman A., Bezpieczeństwo nowoczesnych aplikacji internetowych, 2021

[3] Furtak M., Bezpieczeństwo aplikacji internetowych, Politechnika Lubelska, Instytut Informatyki, Lublin, 2017

[4] Threats and vulnerabilities in web applications 2020–2021 https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020-2021/ (data dostępu 03.01.2024 r.)

[5] Hoiberg S., JavaScript in the industry https://simonhoiberg-ebooks-assets.s3.eu-west-2.amazonaws.com/JavaScript+In+The+Industry.pdf (data dostępu 03.01.2024 r.)

[6] React Introduction https://www.w3schools.com/react/react_intro.asp (data dostępu 14.01.2024 r)

[7] Npm trends https://npmtrends.com/angular-vs-jQuery-vs-react-vs-vue (data dostępu 03.01.2024 r.)

[8] Vercel https://nextjs.org/docs (data dostępu 14.01.2024 r)

[9] Kumar N., Next.js vs React – What are theDifferences?, 2023 (8) https://www.freecodecamp.org/news/next-vs-react/ (data dostępu 03.01.2024 r)

[10] Capała Ł., Skublewska-Paszkowska M., Porównanie szkieletów aplikacji AngularJS i React.js na przykładzie aplikacji internetowej, Politechnika Lubelska, Instytut Informatyki, Lublin, 2017

[11] Minnick C., Building User Interfaces with ReactJS, 2022

[12] Grov M., Building User Interfaces Using Virtual DOM, University of Oslo, Department of Informatics, 2015

[13] Meredova A., Comparison of Server-Side Rendering Capabilities of React and Vue, Haaga-Helia University of Applied Sciences, 2023

[14] Introduction to Identity and Access Management (IAM) https://auth0.com/docs/get-started/identity-fundamentals/identity-and-access-management

[15] Benefits of JWTs https://auth0.com/docs/secure/tokens/json-web-tokens#benefits-of-jwts (data dostępu 05.01.2024 r.)

[16] JSON Web Token Structure https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-structure (data dostepu 05.01.2024 r.)

[17] Introduction to Auth0 https://auth0.com/docs/get-started/identity-fundamentals/introduction-to-auth0#identity-fundamentals (data dostępu 05.01.2024 r.)

[18] Cordella A., Web application penetration testing: an analysis of a corporate application according to OWASP guidelines, University of Bologna, 2018

[19] Alecu F., Cross Site Scripting (XSS) in Action, The Bucharest University of Economic Studies, Department of Economic Informatics and Cybernetics, Romania, 2012

[20] Fejzaj J., Ylli E., Man in the Middle: Attack and Protection, Polytechnic University of Tirana, Faculty of Information Technolog, Albania

[21] Stone P., New attacks against framed web pages, 2010

[22] Lenaerts-Bergmans., Brute Force Attacks https://www.crowdstrike.com/cybersecurity-101/brute-force-attacks/ (data dostępu 05.01.2024 r.)

[23] Loshin P., Open Web Application Security Project (OWASP) https://www.techtarget.com/searchsoftwarequality/definition/OWASP (data dostępu 05.01.2024 r.)

[24] Selvi J., Bypassing HTTP Strict Transport Security https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-wp.pdf (data dostępu 04.01.2024 r.)

[25] Meredith J., Content Security Policy Best Practices https://research.nccgroup.com/wp-content/uploads/2020/07/csp_best_practices.pdf (data dostępu 04.01.2024 r.)

[26] Albeniz Z., Morgenroth S., Yildirimkaya U., HTTP Security Headers, 2018

[27] Bratkowski P., Flaga cookie – HttpOnly https://sekurak.pl/flaga-cookie-httponly/ (data dostępu 05.01.2024 r.)

[28] Piosek M., Flaga cookies SameSite – jak działa i przed czym zapewnia ochronę? https://sekurak.pl/flaga-cookies-samesite-jak-dziala-i-przed-czym-zapewnia-ochrone/ (data dostępu 05.01.2024 r.)

[29] Golovyrin L., Analysis of tools for static security testing of applications, Czech

Technical University in Prague, Faculty of Electrical Engineering Department of Computer Science, 2023

[30] Woźniak J., OWASP ZAP: Opis narzędzia, główne funkcjonalności i przydatne źródła wiedzy https://www.droptica.pl/blog/owasp-zap-opis-narzedzia-glowne-funkcjonalnosci-i-przydatne-zrodla-wiedzy/ (data dostępu 05.01.2024 r.)

[31] https://nextjs.org/docs/pages/api-reference/create-next-app (data dostępu 05.01.2024 r.)

[32] Next.js SDK https://github.com/auth0/nextjs-auth0 (data dostępu 10.2023 r)

[33] Angular url sanitaizer https://github.com/angular/angular/blob/main/packages/core/src/sanitization/url_sanitizer.ts (data dostępu 23.12.2023 r)

[34] Server rendering https://redux.js.org/usage/server-rendering (data dostępu 23.12.2023 r)

[35] CPS https://nextjs.org/docs/app/building-your-application/configuring/content-security-policy (data dostępu 23.12.2023 r)

Spis ilustracji

Spis listingów