Politechnika Krakowska

im. Tadeusza Kościuszki

Wydział Informatyki i Telekomunikacji

Radosław Kraj

Numer albumu: 145691

Testy bezpieczeństwa do autorskiego systemu do przetwarzania plików multimedialnych

Security tests for the original system for processing multimedia files

Praca magisterska

na kierunku informatyka

specjalność Cyberbezpieczeństwo

Praca wykonana pod kierunkiem:

dr Radosław Kycia

Kraków. 2024

SPIS TREŚCI

1. WSTĘP

1.1. Motywacja

Stworzenie dedykowanej internetowej platformy do przesyłania, przechowywania i zarządzania plikami multimedialnymi przez użytkowników, może generować wiele potencjalnych luk bezpieczeństwa, na każdym etapie jej użytkowania. Właściwa implementacja aplikacji, zawierająca dobrze zaprojektowaną architekturę rozwiązania, dobór optymalnych narzędzi, poprawne polityki przechowywania danych wrażliwych, właściwą implementację procesów autoryzacji, zapewniają komfort dalszego rozwoju oprogramowania, jak również zaufanie użytkowników. Bezpieczeństwo aplikacji jest jednym z tych niefunkcjonalnych wymagań aplikacji, o których należy dbać na każdym etapie rozwoju oprogramowania, gdyż brak stosowania właściwych praktyk w tym zakresie, może bardzo negatywnie wpłynąć na odbiór aplikacji.

1.2. Cel pracy

Celem pracy jest implementacja i wdrożenie autorskiego systemu do przetwarzania i zarządzania plikami multimedialnymi, związanymi z oceną pracy sędziego piłkarskiego, a następnie wszechstronne przeprowadzenie testów bezpieczeństwa stworzonej aplikacji.

1.3. Zakres Pracy

Praca obejmuje implementację silnika aplikacji jak i strony graficznej, wdrożenie na serwer, tak aby umożliwić korzystanie z niej zewnętrznym użytkownikom, a następnie przeprowadzenie audytu bezpieczeństwa. Zostały przeprowadzone testy, prowadzące do odkrycia podatności z szerokiego zakresu bezpieczeństwa aplikacji, obejmujące naruszenia sieciowe, aplikacji webowych z zakresu OWASP, czy błędy uwierzytelniania i autoryzacji. Mając dostęp do kodu, szczegółowo została przeprowadzona analiza kodu w kontekście jego bezpieczeństwa i dobrych praktyk w tym zakresie (ang. secure code review).

1.4. Metodyka pracy

Praca magisterska została podzielona na część teoretyczną, mającą wprowadzać do zagadnień zrealizowanych, w części drugiej – praktycznej, w której zostały ujawnione podatności występujące w oprogramowaniu.

Część praktyczna jest podzielona na kilka etapów. Pierwszym z nich jest implementacja aplikacji, wykorzystując do tego języki programowania:

- Java – do implementacji części serwisowej.
- TypeScript – wraz z biblioteką Angular, do implementacji części klienckiej

Kolejnym etapem jest wdrożenie aplikacji na serwer, tak aby umożliwić korzystanie z niej osobom trzecim. Tutaj została wykorzystana usługa chmury AWS – EC2.

W trzecim etapie przeprowadzono weryfikację kodu pod kątem bezpieczeństwa, oceniając błędy występujące w kodzie aplikacji.

Ostatnim etapem części praktycznej jest przeprowadzenie audytu bezpieczeństwa, sprawdzając istnienie podatności w aplikacji, testując ją zarówno przed atakami sieciowymi, jak i tymi na samą warstwę aplikacji.

I. Część teoretyczna

2. Podstawy teoretyczne bezpiecznego procesu wytwarzania oprogramowania

2.1. Wprowadzenie do bezpieczeństwa aplikacji internetowych

Bezpieczeństwo aplikacji internetowych jest elementem, o który należy dbać na każdym etapie wytwarzania oprogramowania. Począwszy od napisania pierwszej linii kodu, przez proces CI/CD (ang. Continues Integration Continues Delivery) [34], wdrażanie aplikacji, aż po ciągłe utrzymywanie jej dostępnej dla użytkowników. Podczas każdego z tych procesów pojawiają się luki, które mogą w mniejszym lub większym stopniu naruszać bezpieczeństwo aplikacji. Nie wszystkie wykryte podatności muszą być natychmiast łatane, najpierw powinny być odpowiednio sklasyfikowane i ocenione. Istnieje wiele metodologii, pozwalających na przeprowadzenie takiego badania. Najbardziej ogólną z nich jest zestawienie prawdopodobieństwa przeprowadzenia ataku, z jego wpływem na aplikację. Bardziej zaawansowany sposób przedstawia Microsoft w modelu DREAD [3] powstałego jako akronim angielskich słów:

- Szkodliwość (ang. damage) - Wpływ ataku na oprogramowania.
- Powtarzalność (ang. reproducibility) - Łatwość wielokrotnego powtórzenia ataku.
- Zdatność (ang. exploitability) – Miara pracy, którą należy włożyć do wykonania ataku.
- Dotknięci użytkownicy – (ang. affected users) – Ilość użytkowników aplikacji, którzy ucierpieli w wyniku ataku.
- Wykrywalność – (ang. discoverability) – Miara tego jak łatwo jest wykryć atak.

Pozwala to na dokładną ocenę danego zagrożenia i tego, kiedy powinno być ono naprawione. Właściwa ocena zagrożeń pozwala również na sklasyfikowanie największych podatności ponad te, których wpływ na aplikację jak i prawdopodobieństwo wykonania są niewielkie. O decyzji czy daną podatność należy naprawiać natychmiast, może również decydować etap jej wykrycia. Szybkość naprawy błędu wykrytego podczas oceny kodu pod kątem bezpieczeństwa będzie często wysoka i niekosztowna. Naprawa błędu wykrytego na późniejszych etapach działania aplikacji, często będzie wymagała więcej pracy, nierzadko dotykając większej ilości komponentów.

2.2. Weryfikacja kodu pod kątem bezpieczeństwa

Nie istnieje bezpieczne oprogramowanie, bez bezpiecznego kodu. Ważnym krokiem do stworzenia bezpiecznej aplikacji jest ocena kodu pod kątem bezpieczeństwa. Polega ona na audycie wytwarzanego kodu najczęściej przez programistę, skupiając się na dobrych praktykach bezpieczeństwa, zawierających w sobie między innymi takie aspekty jak:

- poprawna konfiguracja,
- właściwa walidacja danych,
- poprawna autoryzacja dostępów,
- logowanie interakcji użytkownika.

W większości przypadków weryfikacja kodu pod kątem bezpieczeństwa jest połączeniem pracy człowieka i wykorzystania technologii, takich jak narzędzia do statycznej analizy kodu. Wykorzystanie tylko jednego z tych elementów, może prowadzić do uchybień. W dużych aplikacjach, składających się z wielu komponentów, setek tysięcy linii kodu, korzystających z wielu bibliotek, sam człowiek nie jest w stanie dokładnie zweryfikować każdego fragmentu oprogramowania. Z drugiej strony narzędzia dokonujące statycznej analizy kodu, nie znają kontekstu biznesowego aplikacji, więc często dostarczają błędy, które nie mają wpływu na bezpieczeństwo aplikacji. Jak twierdzą autorzy książki OWASP Code Review Guide v2, analiza kodu pod kątem bezpieczeństwa jest jedną z najbardziej efektywnych technik do identyfikacji potencjalnych incydentów bezpieczeństwa na wstępnym etapie tworzenia systemów, a w połączeniu z automatycznymi testami i automatycznym skanowaniem, znacząco poprawia efektywność weryfikowania bezpieczeństwa oprogramowania. [1]

2.2.1. Ważne aspekty weryfikacji kodu pod kątem bezpieczeństwa

Wykonując weryfikację kodu pod kątem bezpieczeństwa, należy łączyć posiadaną wiedzę z zakresu cyberbezpieczeństwa na temat potencjalnych zagrożeń wpływających na bezpieczeństwo aplikacji, ze znajomością domeny, tak aby właściwie ocenić powierzchnię potencjalnych ataków, czyli wszystkie zasoby IT, które mogą zostać wykorzystane w celu naruszenia bezpieczeństwa aplikacji. Pod uwagę należy brać nie tylko obecny status oprogramowania, ale również potencjalny jego rozwój i zaistnienie podatności w danym miejscu.

Analiza ryzyka

Analizując kod aplikacji podczas jego sprawdzenia, należy zwracać uwagę na aspekty takie jak:

- realny wpływ zagrożenia na bezpieczeństwo aplikacji,
- łatwość zrealizowania podatności,
- ewentualny atak spowoduje błędy w działaniu aplikacji lub dotknie danych.

Oceniając kod może się okazać, że podatność istnieje, ale samo oprogramowanie jest dostępne dla znikomej ilości klientów albo ewentualne wykorzystanie podatności nie powoduje żadnych konsekwencji, a sam koszt naprawy jest wysoki. W takim wypadku należy być świadomym istnienia podatności, natomiast naprawa nie musi być konieczna na tym etapie rozwoju oprogramowania [1].

Kategoryzacja danych

Dane są jednym z najważniejszych aktywów organizacji. Osoba wykonująca przegląd kodu musi brać pod uwagę czy dane są odpowiednio zabezpieczone. Dotyczy to zwłaszcza danych wrażliwych, takich jak: informacje o użytkownikach, numery kont czy kart kredytowych. Właściwa kategoryzacja danych pozwala na określenie czy potencjalny wyciek lub utrata danych, będzie miało realny wpływ na organizację [1].

Ocena dostępu użytkowników

Często pewne elementy oprogramowania powinny być dostępne tylko dla określonej grupy użytkowników. Dokonując przeglądu aplikacji należy upewnić się, że fragmenty kodu odpowiadające za autoryzację, prawidłowo zabezpieczają przed nieautoryzowanym dostępem do tych miejsc. Dotyczy to zarówno nie wymagających uwierzytelnienia, jak również już zalogowanych użytkowników, którzy z różnych przyczyn np. biznesowych mogą mieć dostęp tylko do pewnych określonych funkcjonalności [1].

Analiza architektury i konfiguracji

Obecnie oprogramowania wykorzystują wiele bibliotek, a także zewnętrznych serwisów. Analizując kod należy zwrócić uwagę czy połączenia do innych aplikacji są bezpieczne, jak również to czy dane pochodzące od nich są odpowiednio weryfikowane. Biblioteki pomagające efektywniej i szybciej tworzyć oprogramowanie, również mogą zawierać potencjalne błędy bezpieczeństwa. W tym wypadku, bardzo dobrze sprawdzają się dostępne narzędzia do analizy bibliotek użytych w programie. Sprawdzając kod źródłowy pod tym kątem, należy pamiętać o weryfikacji konfiguracji, o tym czy klucze,

hasła oraz inne wrażliwe dane konfiguracyjne przechowywane są we właściwy sposób [1].

Logowanie i przechwytywanie wyjątków

Logi w aplikacji oraz rzucane wyjątki, potrafią dostarczyć programistom wiedzę o błędach w aplikacji, a także są wskazówką jak szybo je naprawiać. Używane w niewłaściwy sposób mogą z kolei być miejscem wycieku danych. Sprawdzając kod pod kątem bezpieczeństwa, należy upewnić się, czy błąd pokazywany użytkownikowi nie zawiera nadmiarowych informacji np. o haśle używanym przez użytkownika [1].

2.2.2. Analiza kodu pod kątem bezpieczeństwa, a OWASP Top Ten

OWASP „Top Ten" [20] jest listą najbardziej popularnych podatności występujących w aplikacjach webowych, publikowanym przez Open Web Application Security Project (OWASP). Wykonując analizę kodu pod kątem bezpieczeństwa, lista podatności publikowana przez OWASP może być doskonałym przewodnikiem do tego, jak po kolei, sprawdzać potencjalne podatności w kodzie. Każda z nich ma punkty charakterystyczne, na które należy zwracać uwagę, aby kod był bezpieczny i odporny na różnorodne ataki. Książka Code Review Guide [1] obrazuje, jak bazując na wymienionych podatnościach, można krok po kroku przeprowadzić weryfikację kodu pod kątem bezpieczeństwa.

Nieprawidłowa kontrola dostępu

Nieprawidłowa kontrola dostępu (ang. Broken Access Control), oznacza, że aplikacja nieprawidłowo udziela dostępu do poszczególnych funkcji systemu. Obejmuje to przypadki, kiedy niewłaściwie lub w ogóle nie sprawdzane są uprawnienia, jak również te sytuacje, gdy jest proces ten wykonywany jest zbyt późno i atakujący może wykonać niepożądane akcje jeszcze przed autoryzacją. W tym przypadku podczas oceny kodu należy sprawdzać czy:

- każdy punkt wejścia do aplikacji jest właściwie zabezpieczony,
- każde żądanie jest sprawdzane na odpowiednio wczesnym etapie,
- użytkownik nie może wykonać żadnych akcji przed uzyskaniem autoryzacji,
- każdy widok jest właściwie zabezpieczony.

Błędy w kryptografii

Analiza kodu pod kątem błędów w kryptografii (ang. Cryptographic Failures) jest szczególnie istotna wtedy, gdy aplikacja przechowuje dane wrażliwe. Mogą to być

informacje o kartach kredytowych, dane medyczne, hasła, etc. Takie dane powinny być przechowywane pod postacią zaszyfrowaną, przy użyciu bezpiecznych metod np. SHA-256. Zaimplementowanych przy użyciu dedykowanych, sprawdzonych bibliotek, a udostępniane tylko wtedy, gdy właściwa osoba odpyta o nie w odpowiednim momencie. Tworzenie własnych implementacji metod kryptograficznych jest złą praktyką, która najczęściej prowadzi do wielu błędów.

Wstrzyknięcie kodu

Wstrzyknięcie (ang. Injection) polega na wstawieniu przez atakującego złośliwego kodu, pozwalającego na wykonywanie nieprzewidzianych, niebezpiecznych interakcji z aplikacją. Ocena kodu źródłowego pod kątem wstrzyknięć, opiera się głównie na sprawdzeniu wartości wysyłanych przez użytkownika, zwłaszcza pod postacią ciągu znaków (ang. string). Przy ocenie kodu należy pamiętać, że nigdy nie wolno ufać danym wprowadzonym przez użytkownika, a sprawdzający musi upewnić się:

- Czy wykonywane są operacje złączenia wartości tekstowych pochodzących od użytkownika. W większości przypadków mogą one być narażone na ataki typu wstrzyknięcie.
- Czy sprawdzenie poprawności danych przychodzących od użytkowników są wystarczające, włączając w to ich rozmiar, oczekiwane wartości, typ, czy niebezpieczne znaki.
- Czy zapytania SQL budowane są bezpośrednio z wartości wprowadzonych przez użytkownika oraz czy są właściwie parametryzowane.

Sprawdzenie powyższych jest niezbędne. Dostrzeżenie przez atakującego tych operacji w aplikacji, może być bardzo łatwym polem do wykonania wstrzyknięcia.

Błędny projekt techniczny aplikacji

Błędny, niebezpieczny projekt techniczny aplikacji (ang. Insecure Design) jest to podatność, która pojawia się, gdy dostępy do obiektów są nieprawidłowo kontrolowane. Wykorzystując to, atakujący może uzyskać dostęp do obiektu, do którego nie powinien mieć dostępu, np. poprzez manipulację przesyłanymi do aplikacji parametrami.

Podczas oceny kodu należy sprawdzić czy:

- Użytkownicy są autoryzowani do operacji tylko na tych zasobach, do których powinni mieć dostęp.

- Wszędzie tam, gdzie dostęp jest ograniczony, kod źródłowy powinien sprawdzać czy użytkownik wysyłający żądanie, ma odpowiednie prawa do odczytu lub zapisu.

Błędna konfiguracja zabezpieczeń

Błędna konfiguracja zabezpieczeń (ang. Security misconfiguration), może zostać szybko wykryta podczas sprawdzania kodu, natomiast niezbędne są do tego: znajomość wykorzystanej biblioteki odpowiadającej za operacje związane z bezpieczeństwem – jeśli taka została użyta oraz samych zasad bezpiecznego programowania. Sprawdzając kod, należy między innymi:

- Upewnić się, że istnieje konfiguracja dotycząca CORS (ang. Cross-Origin Resource Sharing) [32], CSRF (ang. Cross-Site Request Forgery) [33].
- Jeśli użyty został JWT Token sprawdzić czy klucz jest wystarczająco skomplikowany i ukryty.
- Jeśli hasła przechowywane są w bazie danych, zweryfikować, czy są one właściwie szyfrowane i przetrzymywane, a ich złożoność jest odpowiednia.
- Uprawnienia do wykonywania operacji są przyznane poprawnie.

Podatne oraz przedawnione komponenty

W aspekcie korzystania z przedawnionych i podatnych komponentów (ang. Vulnerable and Outdated Components), sam przegląd kodu źródłowego może nie przynieść rezultatów, chyba że zdecydujemy się na analizę kodu biblioteki. Jest to możliwe tylko w przypadku, gdy będzie ona publiczna. Nawet wtedy najczęściej nie będzie to dobrym, a często bardzo czasochłonnym rozwiązaniem. Przeglądnięcie jednej z nich może być trudne, natomiast zrobienie tego ze wszystkimi istniejącymi bibliotekami, jest niemal niewykonalne. W tym zakresie należy skorzystać z dostępnych narzędzi do skanowania podatności w bibliotekach, jak również śledzić strony monitorujące najnowsze wersje używanych przez aplikacje rozwiązań i na bieżąco je aktualizować.

Błędy identyfikacji i uwierzytelnienia

Uwierzytelnianie i identyfikacja pozwalają na weryfikację użytkownika i jego dostępu do poszczególnych części systemu. Najczęściej proces ten odbywa się przy użyciu nazwy użytkownika i hasła. Programistyczne błędy w zakresie identyfikacji i uwierzytelnienia (ang. Identification and Authentication Failures), mogą prowadzić do

nieautoryzowanych dostępów do zasobów. Podczas sprawdzania kodu należy upewnić się, że:

- Nazwa użytkownika i hasło przesyłane są w sposób zaszyfrowany, tak aby uniemożliwić wykonywanie ataków człowiek w środku (ang. Man in the Middle).
- Wiadomości po nieudanym logowaniu dostarczają tylko te informacje, które są niezbędne. Komunikat o nieudanym logowaniu nie może zawierać informacji o hasłach, jak również nie może być wskazówką dla atakującego, np. wyświetlając informację, że nazwa użytkownika jest właściwa, a błędne jest jedynie hasło. Jest to cenna informacja dla atakującego, mogącego uskutecznić atak siłowy (ang. Brute Force).
- Hasło przy zakładaniu konta użytkownika posiada odpowiednią złożoność – są odpowiednio długie oraz zawierają bogatą kombinację znaków.
- Jeśli hasła przechowywane są w bazie danych to wykorzystana technika szyfrowania jest bezpieczna.

Błędy w integralności oprogramowania i danych

Błędy w integralności oprogramowania i danych (ang. Software and Data Integrity Failures) często aplikowane są do niewłaściwie zaimplementowanych procesów CI/CD. Podczas weryfikacji kodu w tym obszarze, należy upewnić się, że wykorzystywane biblioteki pochodzą z zaufanych repozytoriów, jak również to czy biblioteki z kodem źródłowym i baza danych jest zabezpieczona przed nieautoryzowanym dostępem.

Błędy w logowaniu i monitorowaniu

Podatność ta odnosi się do wad w logowaniu oraz monitoringu oprogramowania (ang. Security Logging and Monitoring Failures) i scala w sobie przypadki, w których te części aplikacji nie działają poprawnie. Obejmują niewystarczające, wybiórcze komunikaty, przedstawiane w sposób niejasny lub też z opóźnieniem. Wszystkie te czynniki sprawiają, że reakcja na zrealizowane zagrożenie może być spóźniona, a sam atak trudny do wykrycia. Podczas oceany kodu, należy upewnić się, że logi zwracają klarowne informacje, mogą być łatwo filtrowane i stosowane we wszystkich kluczowych komponentach aplikacji.

Fałszowanie żądań po stronie serwera

Fałszowanie żądań po stronie serwera (ang. Server Side Request Forgery) dotyczy głównie serwerów aplikacyjnych, jednak często wykorzystuje urządzenie, które

najczęściej jest jednym z zaufanych w sieci wewnętrznej do wysyłania dowolnych żądań do serwera. Jeśli oprogramowanie pozwala użytkownikom na wprowadzenie dowolnego adresu URL lub pliku bez żadnej walidacji, może okazać się, że będzie sformatowany on w taki sposób, aby wykonać żądania na serwerze. W kodzie aplikacji podczas oceny kodu należy zwrócić szczególną uwagę czy adresy lub pliki przesyłane przez użytkowników są poprawnie sprawdzane, mają oczekiwane rozszerzenia, a ich nazwy lub adresy są zgodne z założeniami.

2.2.3. Statyczna analiza kodu

Narzędzia do statycznej analizy kodu, są przydatnym narzędziem, zwłaszcza gdy oprogramowanie staje się coraz większe i nie jest możliwym całkowity dokładny przegląd aplikacji. Pozwalają na wychwycenie popularnych błędów, które mogą być przeoczone przez człowieka. Umożliwiają przegląd bezpieczeństwa nie tylko kodu źródłowego, ale również dla wykorzystanych bibliotek. Szczególnie warte docenienia jest to przy narzędziach wykorzystujących zależności przechodnie (ang. transitive dependency), takich jak maven [2], czy gradle [20], które przy skorzystaniu z wybranej biblioteki, pobierają inne przez nią wykorzystywane, itd. Bez narzędzi do statycznej analizy kodu, wykrycie zagrożeń sięgających tak daleko, byłoby niemożliwe. Dodatkowo narzędzia te dostarczają raport oraz ranking zagrożeń pomagając w ocenie, które z nich mogą mieć realny wpływ na aplikację. Bazując na dokumentacji narzędzia Snyk [16], który jest platformą umożliwiającą skanowanie, ocenę oraz naprawę podatności występujących w kodzie. Pozwalającą również na monitorowanie jakości kodu, jak również zagrożeń znajdujących się bezpośrednio w nim, a także we wszystkich bibliotekach wykorzystywanych w projekcie, można określić główne aspekty, które brane są pod uwagę podczas określania wyniku podatności. Zgodnie z dokumentacją Snyk. Są to [19]:

- Poziom dotkliwości, obliczany przy wykorzystaniu punktacji CVSS (ang. Common Vulnerability Scoring System) [35] dla danej podatności.
- Dojrzałość powszechnie dostępnych ataków, określane przez zespoły bezpieczeństwa, testujące podatność.
- Łatwość naprawy, weryfikująca czy istnieje prosta metoda naprawy, taka jak aktualizacja wersji oprogramowania, czy programiści muszą poprawić kod samodzielnie.

- Czas istnienia podatności, nowe luki uznawane są za te, o podniesionym ryzyku, przez co również wyżej ocenianie.
- Trendy społeczne, obliczane na podstawie wzmianek o podatności w mediach społecznościowych.
- Pochodzenie ze złośliwych paczek, sprawdzające pochodzenie luk.

Należy jednak pamiętać, że narzędzia do statycznej analizy, skupiają się głównie na kodzie, nie uwzględniając takich aspektów, jak logika biznesowa, walidacje, potencjalnie wrażliwe dane, etc. Dodatkowo mogą generować wyniki określane jako fałszywie pozytywne (ang. false positive), które nie są realnymi zagrożeniami.

2.3. Atak DoS

Zgodnie ze słownikiem terminów z zakresu bezpieczeństwa narodowego [5], blokada usług (ang. Denial of Service – DoS) jest atakiem komputerowym polegającym na zablokowaniu dostępu do systemu, poprzez zajęcie jego zasobów. Polega na nieustannym wysyłaniu dużej ilości pakietów danego typu na adres atakowanego serwisu lub żądań o dużych rozmiarach wypełniających całą dostępną pamięć, powodując brak możliwości obsługi kolejnych zapytań, a tym uniemożliwienie użytkownikom dostępu do usługi. Jedną z metod ataków DoS jest posługiwanie się bardzo dużymi plikami lub bardzo dużą ilością plików, które w łatwy sposób przeciążają serwer. Podatne są na to aplikacje, które nie sprawdzają rozmiarów przesyłanych plików we właściwy sposób. Ataki typu DoS są łatwe do wykrycia. Najczęściej na takie próby mogą wskazywać duże ilości żądań pochodzące z określonego adresu IP, nagły wzrost żądań do danego punktu końcowego czy niestandardowe wzrosty ruchu. Przy odpowiednim monitoringu aplikacji, atakującego można bardzo szybko wykryć, a następnie uniemożliwić mu dostęp do serwisu przy użyciu firewall.

Zdecydowanie trudniejszą do wykrycia i odparcia formą ataku DoS jest DDoS - rozproszona blokada usług (ang. Distributed Denial of Service), wykorzystujący w tym celu wiele zainfekowanych komputerów, zazwyczaj koordynowanych z jednego miejsca tzw. botnet. W takim przypadku na atak często będą wskazywać nietypowe wzorce ruchu np. nagły, nieregularny wzrost ilości zapytań.

2.4. Ataki na uwierzytelnianie

Proces uwierzytelniania obecny jest w niemal każdej aplikacji przetwarzającej dane. Mogą być to informacje medyczne, osobiste, finansowe lub jakiekolwiek inne,

które nie powinny być widoczne dla każdego użytkownika Internetu. W celu uzyskania dostępu do tych danych, użytkownik musi poświadczyć w procesie logowania, a następnie autoryzacji, że może otrzymać on prawo do ich przeglądania czy modyfikacji. Proces uwierzytelniania najczęściej polega na podaniu nazwy użytkownika oraz hasła, a operacja kończy się sukcesem, jeśli podane dane są zgodne z tymi, które są przechowywane w systemie. Kolejnym etapem jest autoryzacja, odpowiadająca za weryfikację do jakich danych użytkownik ma dostęp oraz to jakie operacje może wykonywać. O ile logowanie odbywa się raz na zależny od metody uwierzytelniania okres czasu, to autoryzacja powinna być wykonywana przy każdym odpytaniu serwera. Najpopularniejszymi metodami uwierzytelniania użytkowników w aplikacjach są:

- JWT Token
- Oauth2
- Uwierzytelnianie wieloetapowe
- API Key Authentication
- Uwierzytelnianie biometryczne

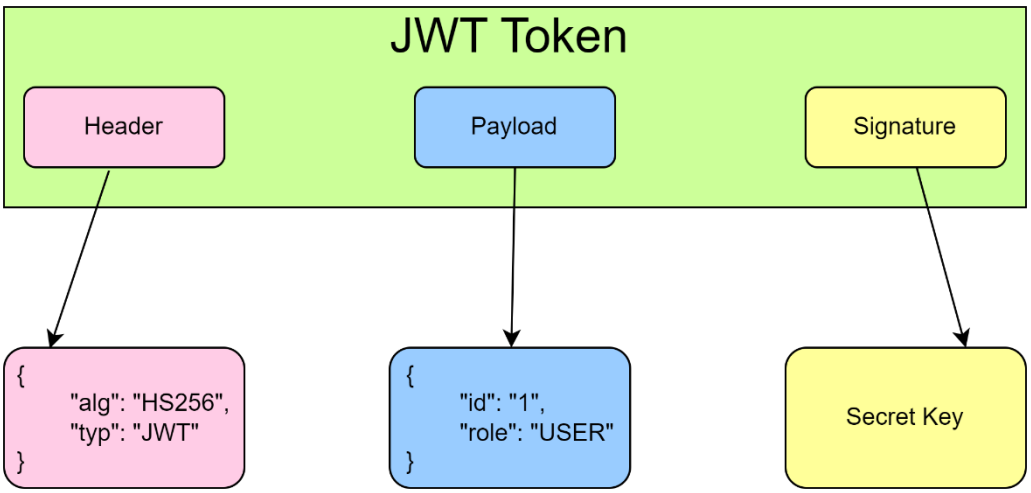
Ataki, w których atakującemu udaje się uzyskać nieautoryzowany dostęp do przejętego konta, otrzymuje pełnię jego danych, jak również możliwość wykonywania operacji w imieniu ofiary. Wykorzystywanie podatności w procesie uwierzytelnienia i autoryzacji jest bardzo popularne zarówno z winy użytkowników, którzy często zapominają o tym jak ważne jest przestrzeganie zasad bezpieczeństwa, jak również twórców aplikacji, którzy z braku, czasu, lub wiedzy, również nie są w stanie odpowiednio przygotować aplikacji na ataki.

2.4.1. JWT Token

Jest jednym z najpopularniejszych standardów określających sposób bezpiecznego przesyłania informacji w formacie JSON (ang, JavaScript Object Notation). Transportowane w ten sposób informacje są podpisane lub zaszyfrowane cyfrowo, przy użyciu wybranego algorytmu szyfrowania takiego jak HMAC, czy RSA. Klucz JWT składa się z szeregu deklaracji występujących w parach klucz-wartość, podzielonych na trzy sekcje (rysunek 2.1):

- Nagłówek (ang. header) – Informuje o typie tokenu oraz zastosowanym algorytmie szyfrowania.

- Ładunek (ang. payload) – Zawiera informacje, które jedna ze stron chce przekazać.
- Podpis (ang. signature) – Używany do weryfikacji tokenu, szyfrowany wybranym algorytmem i przy użyciu klucza.



Rysunek 2.1. Schemat budowy Tokena JWT.
Źródło: Opracowanie własne na podstawie [7]

Zagrożenia w korzystaniu z JWT Token

Dobrze zaimplementowany token JWT w większości przypadków jest bezpiecznym i skutecznym rozwiązaniem. Pomimo to, istnieją jednak możliwości jego złamania i uzyskania nieautoryzowanego dostępu do aplikacji.

Jedną z nich jest wykorzystanie „none” jako algorytmu szyfrowania w nagłówku tokena. Użycie go powoduje, że algorytm nie będzie weryfikował wykorzystanego szyfrowania, a tym samym uznawał brak podpisu za właściwe, oczekiwane zachowanie. Niestety standard JWT pozwala na takie rozwiązanie, co przy niezmienionej sekcji z ładunkiem i pustej sekcji podpisu, pozwala na uzyskanie nieautoryzowanego dostępu do aplikacji.

Kolejnym problemem dla twórców aplikacji wykorzystujących token do autoryzacji, może być próba łamania klucza tokenu. W tym przypadku wystarczy uzyskać dowolny token – aby to zrobić często wystarczy zarejestrować się do aplikacji i wykonać żądanie, a następnie przy pomocy narzędzi takich jak ettercap [14], próbować uzyskać klucz podpisu. Jak podają twórcy książki Bezpieczeństwo Aplikacji Webowych [9], przy użyciu kliku mocnych kart graficznych atakujący jest w stanie wykonać powyżej miliarda sprawdzeń na sekundę. Przy słabym podpisie, pozwala to na szybkie uzyskanie klucza, a następnie wykonywanie dowolnych operacji na serwerze.

Jednym z ułatwień dla atakującego jest wykorzystanie tokena w każdym zapytaniu do serwera, co sprawia, że istnieje bardzo dużo szans na przechwycenie i wykorzystanie go. W takim przypadku atakujący może wykonywać żądania w imieniu użytkownika. Wszystkie te podatności wynikają z prostych błędów programistycznych, takich jak pozwolenie na użycie dowolnego algorytmu szyfrowania, zbyt słaby klucz czy używanie złego protokołu komunikacji, natomiast powodują, że uzyskanie nieautoryzowanego dostępu, jest możliwe.

2.4.2. Atak siłowy

Atak siłowy (ang. Brute Force) polega na wielokrotnych próbach zestawienia ze sobą nazwy użytkownika i hasła, aż do uzyskania właściwej kombinacji, w celu nieautoryzowanego dostępu do konta użytkownika. Jest dość prostą metodą, cechującą się wysoką złożonością obliczeniową, co sprawia, że wraz ze wzrostem długości i skomplikowania hasła, wliczając w to ilość możliwych do wykorzystania znaków, skuteczność takiego ataku maleje. Brak wymagań co do hasła lub używanie w nim tylko jednego typu znaków np. małych liter, znacząco osłabia jego siłę. Maksymalną długość łamania hasła tą metodą można określić jako $x^{y,}$ gdzie x to ilość możliwych do użycia znaków, a y długość hasła. W celu zapobiegnięcia takim atakom, należy wprowadzić w aplikacji czasowe ograniczenie logowania po określonej ilości nieudanych prób lub mechanizmy Captcha. Tworząc aplikację, należy również wymagać od użytkowników złożonych haseł, składających się z małych i dużych liter, liczb oraz znaków niestandardowych. Przykładowo hasło, które składa się z minimum ośmiu znaków, w tym jednej cyfry, zgodnie z grafiką poniżej (rysunek 2.2) przy ataku siłowym, nie będzie łamane dłużej niż dwie minuty.

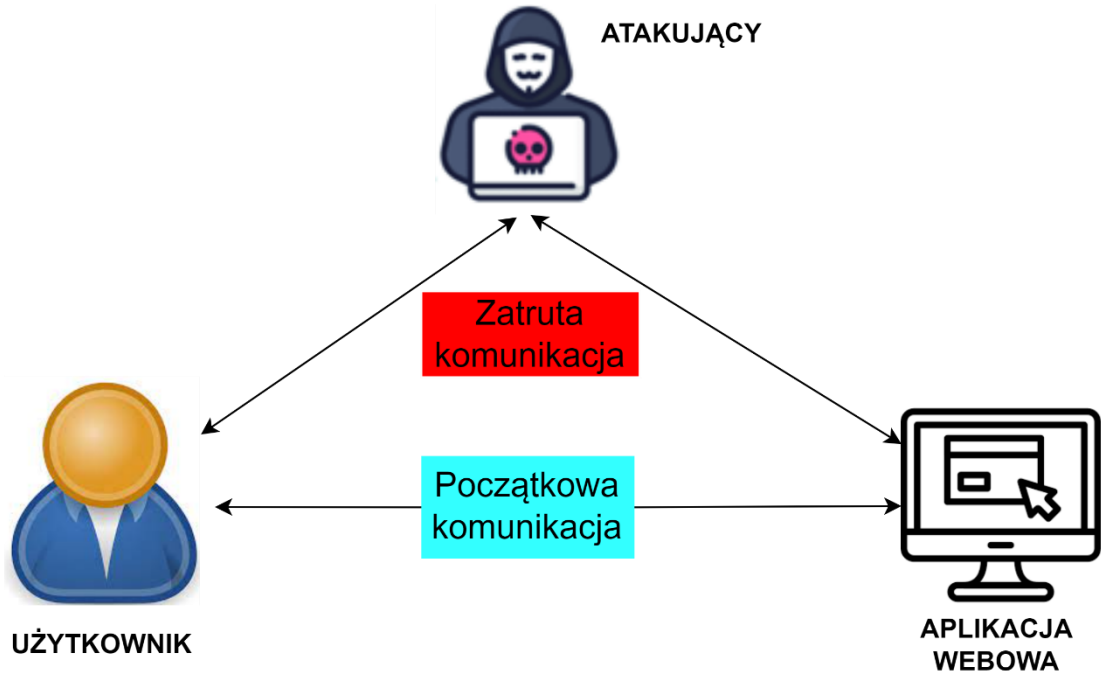
Ilość znaków	Liczby	Małe litery	Małe i wielkie litery	Liczby, małe i wielkie litery	Liczby, małe i wielkie litery, symbole
4	Natychmiast	Natychmiast	Natychmiast	Natychmiast	Natychmiast
5	Natychmiast	Natychmiast	Natychmiast	Natychmiast	Natychmiast
6	Natychmiast	Natychmiast	Natychmiast	Natychmiast	Natychmiast
7	Natychmiast	Natychmiast	1 sekunda	2 sekundy	4 sekundy
8	Natychmiast	Natychmiast	28 sekund	2 minuty	5 minut
9	Natychmiast	3 sekundy	24 minuty	2 godziny	6 godzin
10	Natychmiast	1 minuta	21 godzin	5 dni	2 tygodnie
11	Natychmiast	32 minuty	1 miesiąc	10 miesięcy	3 lata
12	1 sekunda	14 godzin	6 lat	53 lata	226 lat

Rysunek 2.2. Tabela długości łamania hasła w zależności od jego złożoności.
Źródło: Opracowanie własne na podstawie [6]

Zgodnie z powyższym obrazem (rysunek 2.2.), hasła użytkowników powinny być bezpieczne, jeśli będą składać się z minimum 11 znaków, przy użyciu kombinacji symboli i znaków.

2.5. Man in the Middle

Człowiek w środku jest atakiem, w którym napastnik chce przejąć żądanie przesyłane pomiędzy użytkownikiem, a serwerem. Jednym ze sposobów, aby to uczynić jest zatrucie połączenia przez atakującego, tak aby komunikacja nie odbywała się bezpośrednio od ofiary do serwisu, ale aby przechodziła przez napastnika, który dopiero wtedy przekaże ją do właściwego odbiorcy. Użytkownik sieci, może być całkowicie nieświadomy, że cały ruch pomiędzy nim a internatem, odbywa się przez atakującego, który podsłuchuje ruch. Graficzną wizualizację przepływu ataku człowiek w środku, zaprezentowano na rysunku 2.3.



Rysunek 2.3. Zestawienie komunikacji HTTP podczas ataku Man in The Middle, z komunikacją przebiegającą we właściwy sposób.
Źródło: Opracowanie własne na podstawie [12]

W celu wykonania żądania, użytkownik musi znać adres IP i adres MAC bramy (ang. gateway) - najczęściej jest to router wifi. Informacje o powiązaniach adresu IP i MAC przechowywane są w tablicy ARP. Atakujący, aby przechwytywać ruch wysyła do użytkownika informację, że adres IP routera jest powiązany z jego adresem MAC, a do routera informacje, która wiąże adres IP użytkownika również z adresem MAC atakującego, zatruwając tablicę ARP. Wtedy cały ruch odbywa już się poprzez atakującego i może on z użyciem narzędzi takich jak np. Wireshark, podglądać wysyłane żądania. Jeśli użytkownik korzysta ze stron używających szyfrowania, w większości przypadków atak będzie nieskuteczny. Natomiast jeśli dane zostały przesłane nieszyfrowanym protokołem np. HTTP, wtedy atakujący może podglądnąć wszystkie informacje, wliczając w to dane wrażliwe, takie jak nazwę użytkownika, hasło, a także odpowiedź serwera na wysłane żądanie.

2.6. Niebezpieczne bezpośrednie odwołanie do obiektu

Niebezpieczne bezpośrednie odwołanie do obiektu (ang, Insecure Direct Object Reference) – IDOR, jest podatnością będącą podzbiorem najpopularniejszej podatności OWASP Top 10 [20], Broken Access Control. Opierają się na błędach w implementacji logiki biznesowej, błędnie weryfikującą prawa dostępu do obiektów. Pozwala na dostęp lub modyfikacje danych, do których atakujący nie powinien mieć uprawnień. Podatność

pojawia się najczęściej, gdy programiści bezwiednie ufają danym pochodzącym od użytkowników i na ich podstawie przydzielają dostęp do obiektów, takich jak pliki, encje, etc.

Podatność IDOR w aplikacji łatwo zweryfikować wykorzystując oprogramowanie takie jak np. Burp Suite [17]. Jeśli aplikacja przyznaje użytkownikowi dostęp do obiektu wykorzystując konkretne dane, np. id obiektu, chcąc wykonać atak, wystarczy wykonać to żądanie wysyłając id innego istniejącego obiektu. Kiedy odpowiedź od serwera będzie pozytywna i zostanie zwrócony inny obiekt, do którego atakujący nie miał uprawnień. Oznacza to, że podatność istnieje.

2.7. Błędy w konfiguracji

Błędy w konfiguracji (ang. Security Misconfiguration) jest stałym punktem na liście OWASP Top 10 [20] i zrzesza w sobie podatności wynikające ze złej konfiguracji bezpieczeństwa. Wliczają się w to wszelkie aspekty dotyczące bezpieczeństwa sieci, takie jak: otwarcie nieużywanych portów czy usług, brak utwardzenia usług sieciowych, itd. Dotyczą one również części aplikacyjnej i zawierają w sobie stosowanie kont o domyślnych nazwach użytkowników i hasłach, błędną konfigurację serwerów aplikacyjnych, czy wyświetlanie informacji o błędach, zawierające zbyt wiele detali, np. informując atakujących o istnieniu użytkowników o podanej nazwie. Kolejne przykłady podają twórcy blogu Sekurak [18], pokazując jak zbyt obszerne wyświetlanie treści wyjątków może prowadzić do ujawnienia informacji o języku programowania, w którym napisana jest aplikacja. Prezentują również wykorzystane biblioteki w aplikacjach, konkretne ścieżki do plików na serwerze czy informacje o wykorzystanej bazie danych. Znając dokładną wersję komponentów aplikacji, atakujący może wyszukać znane podatności znalezione dla niej i w ten sposób dokonać prostego ataku na oprogramowanie.

II. Część praktyczna

3. Opis aplikacji i analiza bezpieczeństwa oprogramowania

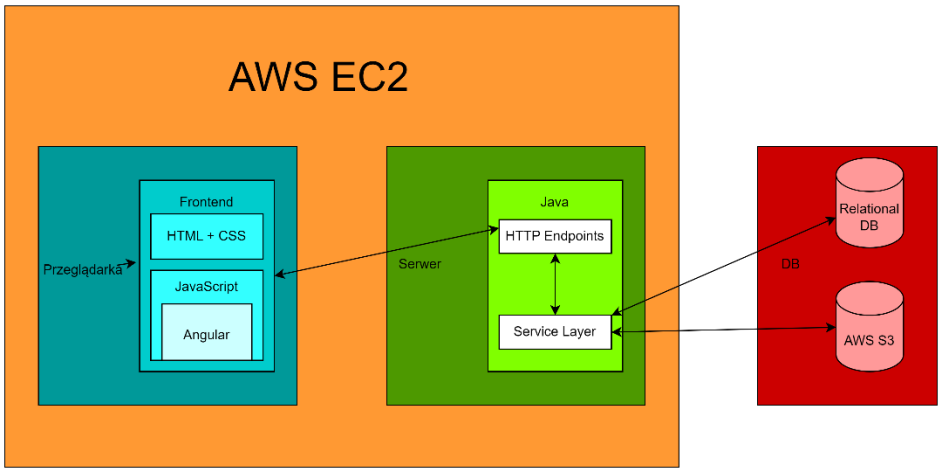
3.1. Opis aplikacji

Stworzona, a następnie przetestowana w ramach niniejszej pracy dyplomowej aplikacja internetowa została nazwana „4referee". Zaprojektowana, zaimplementowana, a następnie poddana audytowi bezpieczeństwa przez autora pracy. Jej zadaniem jest przyśpieszenie procesu oceny video sędziów piłkarskich, poprzez gromadzenie plików multimedialnych zawierających kontrowersyjne sytuacje meczowe, z prowadzonych przez arbitrów meczów. Ułatwia ona również współdzielenie plików video, ulepszając proces edukacji sędziów piłkarskich. Większość funkcjonalności aplikacji dostępna jest tylko po uwierzytelnieniu. Użytkownicy niezalogowani mogą korzystać jedynie z panelu do logowania oraz rejestracji. Po pomyślnym zalogowaniu się do systemu, użytkownicy mogą w zależności od pełnionej roli w systemie dodawać, przeglądać, usuwać pliki multimedialne, sprawdzać dane o sobie lub innych użytkownikach.

Aplikacja po stronie serwerowej została napisana w języku Java, szeroko wykorzystując bibliotekę Spring. Do obsługi relacyjnej bazy danych, zastosowane zostało narzędzie Spring Data, do zapewnienia bezpiecznych procesów uwierzytelniania i autoryzacji Spring Security, a za przetwarzanie JWT Token odpowiada biblioteka jjwt.

Za kliencką część aplikacji odpowiadają HTML, CSS oraz TypeScript, który jest wykorzystywany przez Angular. Komunikacja pomiędzy obiema częściami aplikacji odbywa się z wykorzystaniem protokołu REST.

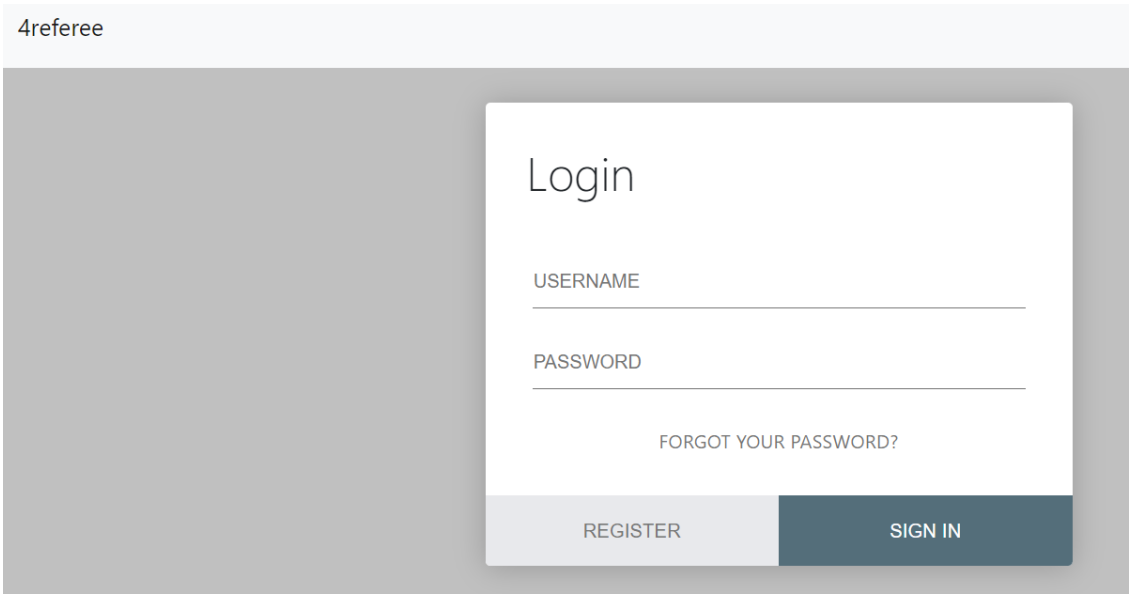
Oprogramowanie korzysta również z usług chmury AWS, dostarczanej przez Amazon. Do udostępnienia aplikacji wykorzystywana jest usługa EC2 [30], gdzie przy pomocy Docker [29] serwis jest uruchamiany. Kolejną usługą AWS, z której korzysta oprogramowanie jest S3 [31], która służy do przechowywania dużych plików, umożliwiając swobodne zapisywanie i odczytywanie materiałów multimedialnych. Całość architektury widoczna jest na rysunku 3.1.



Rysunek 3.1. Diagram przedstawiający architekturę aplikacji.
Źródło: opracowanie własne

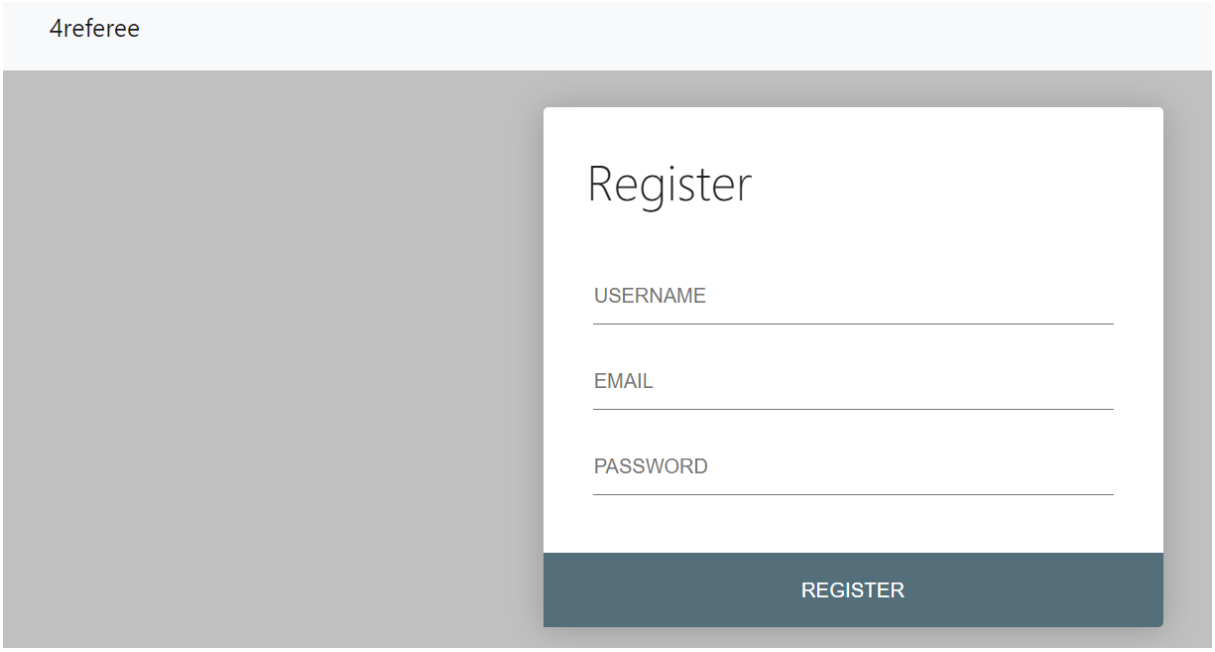
Rodzaj przeprowadzonego testu penetracyjnego, określony jest terminem białej skrzynki (ang. white box). Oznacza to, że podczas testowania dostępne są wszystkie informacje o systemie i jego szczegółach. Zalicza się do tego dostęp do kodu źródłowego, architektury, infrastruktury, przypadków biznesowych i wszystkich innych danych, będących wartościowymi podczas testów bezpieczeństwa.

Po wejściu na stronę aplikacji tak jak zostało to zaprezentowane na rysunku 3.2, widoczne są jedyne akcje dostępne dla nieautoryzowanego użytkownika: logowanie i rejestracja. Uwierzytelnianie odbywa się przy użyciu nazwy użytkownika oraz hasła. Testując tą część warto zwrócić uwagę na zabezpieczania jakie zostały użyte po serii nieudanych logowań, czy błędy wyświetlane podczas przekazania błędnych danych.



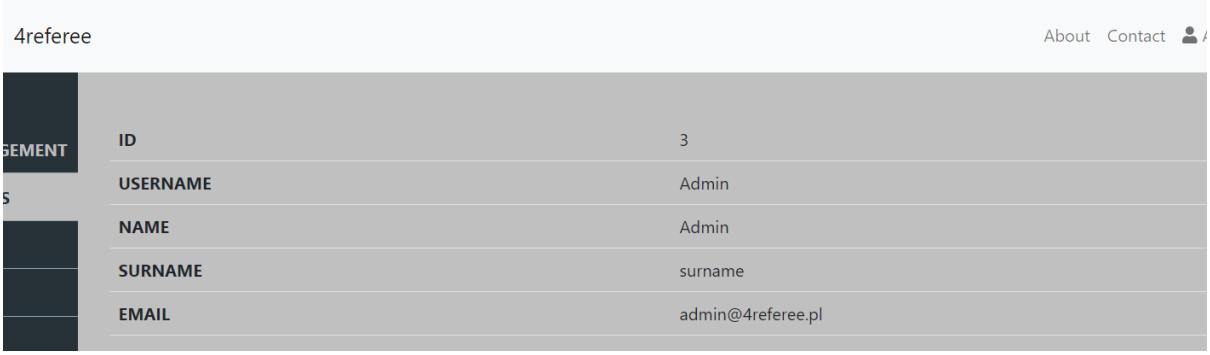
Rysunek 3.2. Panel logowania do aplikacji
Źródło: opracowanie własne

Panel rejestracji (rysunek 3.3) zawiera tylko trzy pola, nazwę użytkownika, hasło oraz adres email, na który zostanie wysłany link potwierdzający rejestrację i aktywowanie konta użytkownika. Tutaj należy sprawdzić wdrożoną politykę haseł aplikacji, jeśli taka istnieje, informacje zwrotne o niepoprawnych danych wprowadzonych przez użytkownika czy proces aktywowania kont użytkowników.



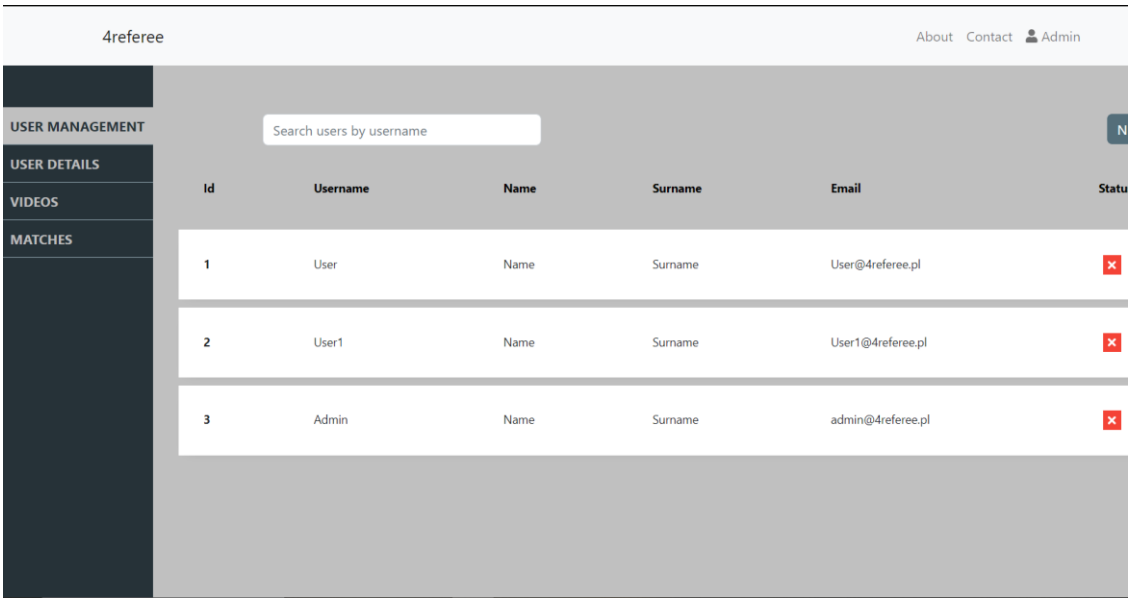
Rysunek 3.3. Panel rejestracji do aplikacji.
Źródło: opracowanie własne

Po zalogowaniu się użytkownik zostaje przekierowany na stronę zwierającą informację o nim, przedstawione na rysunku 3.4. Użytkownik nie posiada możliwości modyfikowania swoich danych.



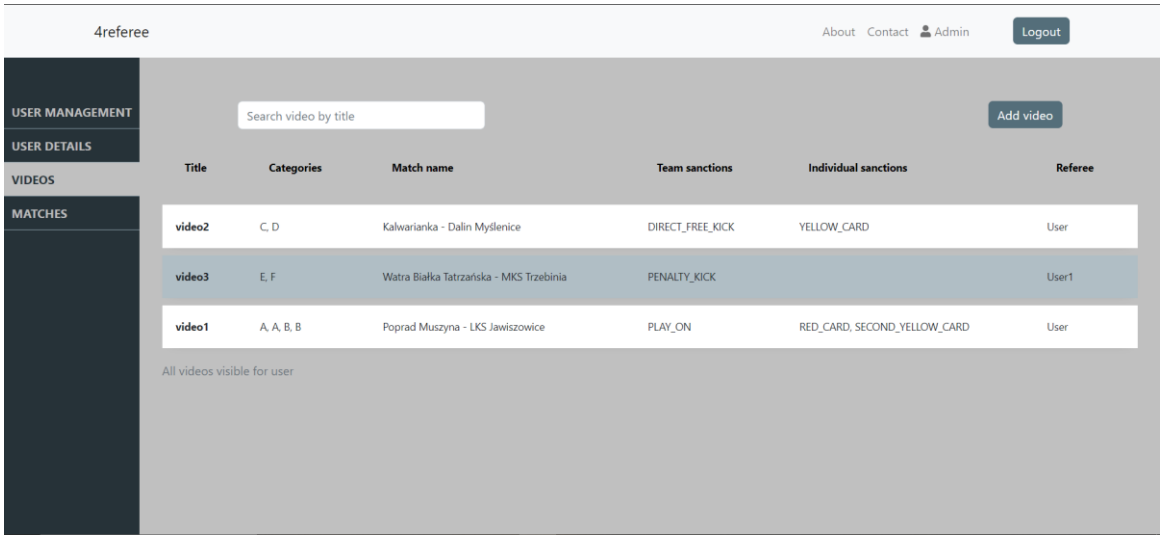
Rysunek 3.4. Strona zawierająca informacje o użytkowniku.
Źródło: opracowanie własne

W zależności od posiadanej roli w systemie, osoba korzystająca z aplikacji nie posiada lub posiada - jeśli jest administratorem, dostęp do panelu z informacją o wszystkich użytkownikach systemu, wraz z możliwością ich usuwania, tak jak jest to widoczne na rysunku 3.5.



Rysunek 3.5. Strona administratora, do zarządzania wszystkimi użytkownikami w systemie.
Źródło: opracowanie własne

Klienci aplikacji mają dostęp do modułu video, w którym w zależności od pełnionej funkcji mogą widzieć tylko swoje pliki – dla zwykłych użytkowników, lub materiały wszystkich osób, jeśli pełnią rolę administratora. Każdy plik posiada swój unikalny tytuł, kategorię, nazwę meczu, z którego pochodzi oraz ewentualne sankcje. Zostało to pokazane na rysunku 3.6.

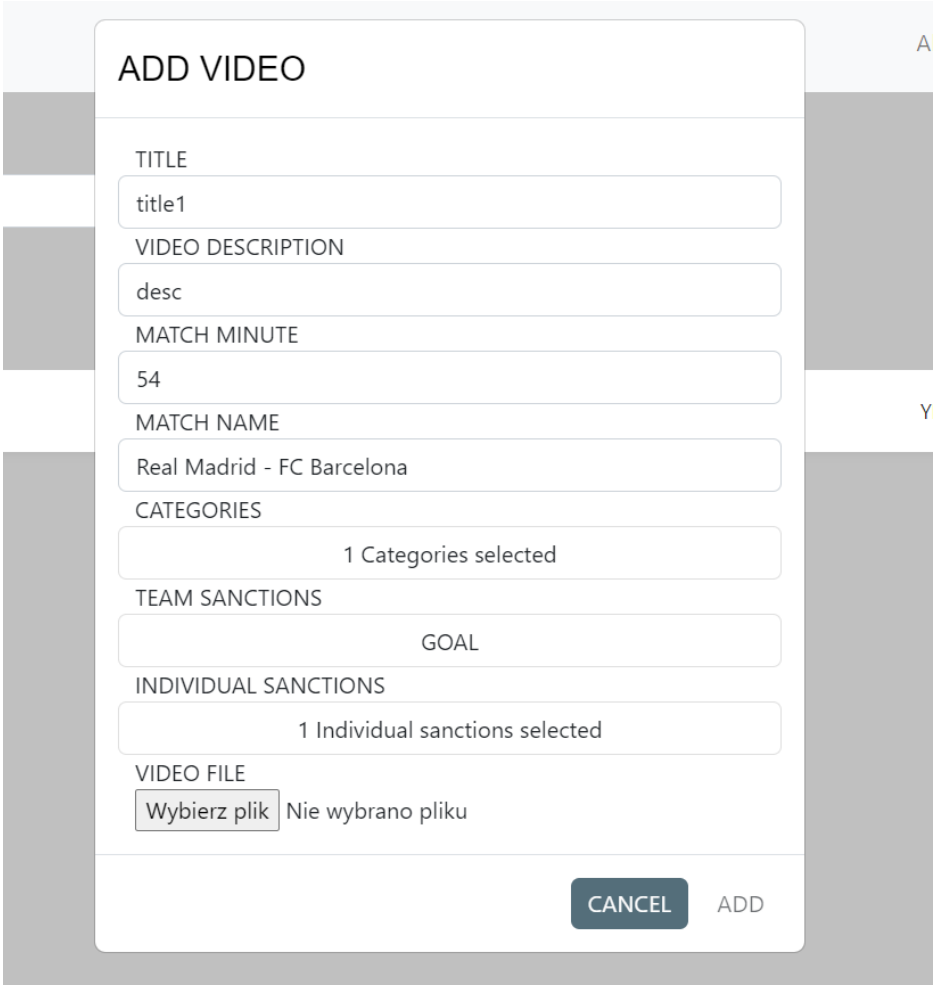


Rysunek 3.6. Strona z dostępnymi plikami multimedialnymi dla użytkownika.
Źródło: opracowanie własne

Każdy plik może być wyszukany po fragmencie tytułu. Wynik działania tej funkcjonalności widoczny jest na rysunku 3.7.

Rysunek 3.7. Filtrowanie plików multimedialnych po tytule.
Źródło: opracowanie własne

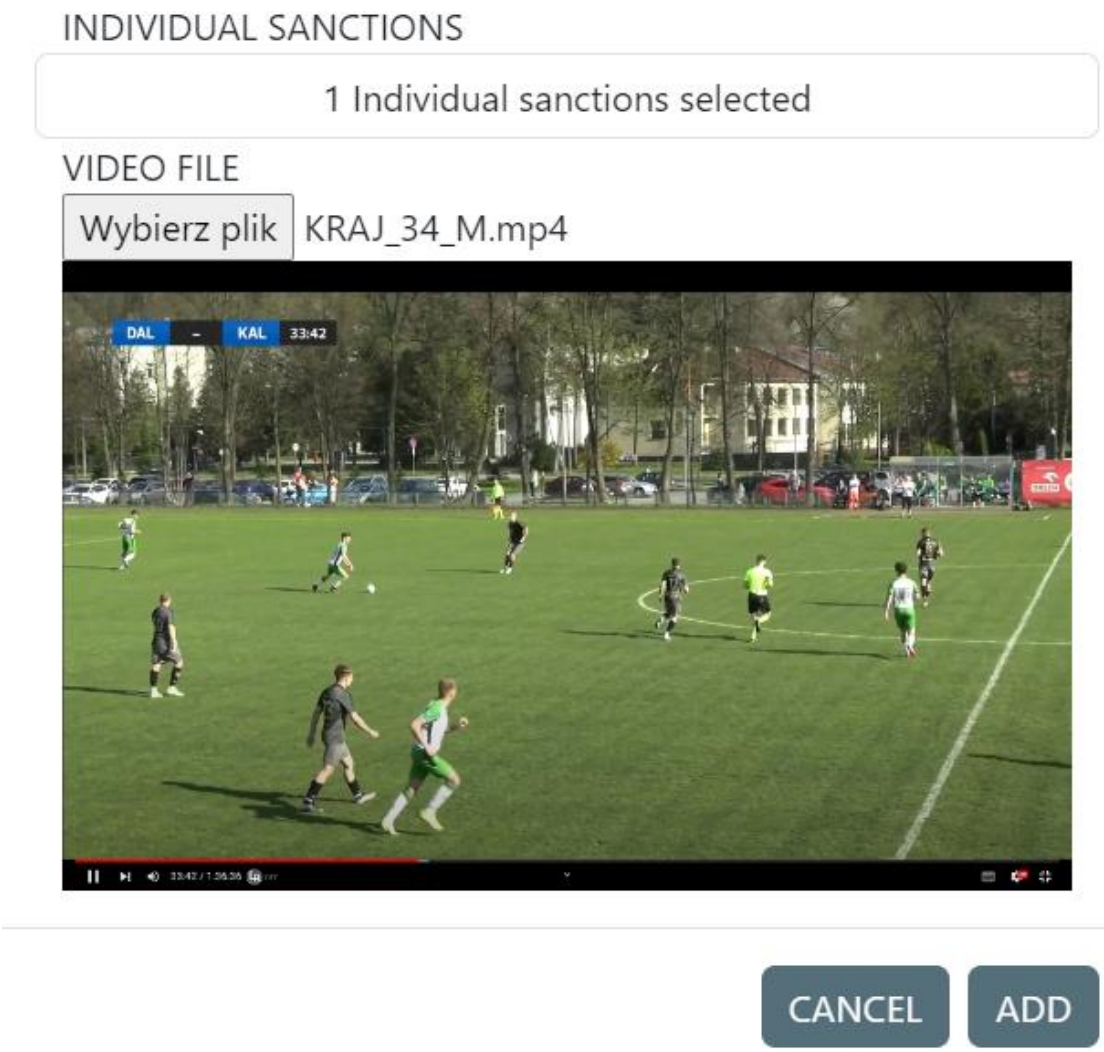
Miejscem występowania wielu potencjalnych podatności, jest również panel umożliwiający dodawanie plików (rysunek 3.8). Zawiera on wszystkie dane dotyczące video. Tutaj należy dokładnie sprawdzać wszelkie dane pochodzące od użytkownika.



Rysunek 3.8. Panel do dodawania plików przez użytkownika wraz z jego detalami.
Źródło: opracowanie własne

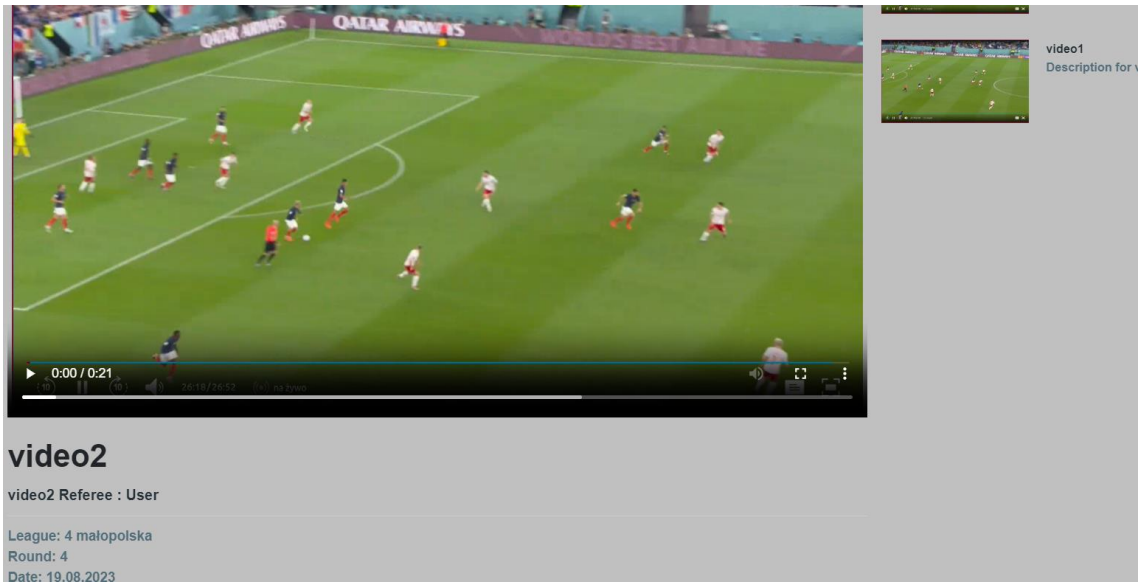
Po wgraniu pliku video automatycznie generowany jest obraz z wybranym fragment z dodanego nagrania, widoczny na rysunku 3.9. Pozwala upewnić się czy został

wybrany właściwy plik. Podczas dodawania obu plików, należy dokonać dokładnego sprawdzenia, czy są one odpowiednio walidowane przez stronę serwerową i odporne na wszelkiego rodzaju ataki.



Rysunek 3.9. Generowanie obrazu po dodaniu pliku multimedialnego przez użytkownika.
Źródło: opracowanie własne

Po wybraniu dowolnego z dostępnych do podglądu plików, użytkownik zostaje przekierowany na stronę, widoczną na rysunku 3.10, gdzie wyświetlany jest duży odtwarzacz z wybranym nagraniem, a także pasek z sugerowanymi, dostępnymi dla niego plikami multimedialnymi.



Rysunek 3.10. Strona umożliwiająca oglądanie plików multimedialnych.
Źródło: opracowanie własne

3.1.1. Narzędzia wykorzystane do implementacji aplikacji

H2

Baza H2 jest lekką i szybką relacyjną bazą danych napisaną i wykorzystywaną głównie w języku Java. Często używana podczas testów integracyjnych lub początkowych faz implementacji rozwiązań. Została wybrana jako początkowe, przejściowe, wbudowane rozwiązanie, ze względu na wygodne użycie w połączeniu z biblioteką Spring oraz łatwość konfiguracji [24].

Spring Framework

Spring jest biblioteką znacznie przyśpieszającą tworzenie aplikacji internetowych pisanych w języku Java. Umożliwiając łatwą obsługę wstrzykiwania zależności, obsługę transakcji, tworzenia testów czy serwera aplikacyjnego [25].

Spring Boot

Spring Boot jest projektem rozwijanym w obrębie biblioteki Spring, ułatwiającym szybkie i proste tworzenie aplikacji internetowych, znacząco upraszczając konfigurację środowiska serwerowego [26].

Spring Data

Spring Data jest biblioteką pozwalającą na łatwą obsługę wszelkich interakcji z bazą danych. Pozwala na automatyczne generowanie zapytań i zapewnienie wstrzykiwania parametrycznego, które zapewnia, że dane pochodzące od użytkownika są przekazywane do zapytań SQL jako parametry, a nie przy pomocy łączenia wartości typu string. Dzięki temu zrealizowanie ataku SQL jest ograniczone do minimum [27].

Spring Security

Spring Security jest projektem dostarczającym narzędzia do implementacji i zarządzania bezpieczeństwem w aplikacji. Dostarcza szereg możliwości uwierzytelniania użytkownika za pomocą różnych źródeł, LDAP, OAuth, JWT itd. Obsługuje mechanizmy autoryzacji oparte na rolach, a także uprawnieniach użytkowników. Zapewnia ochronę przed atakami takimi jak CSRF (Cross-site request forgery), XSS (Cross-site scripting), atakami na sesję, uwierzytelnienie etc. [28].

Docker

Docker jest narzędziem umożliwiającym konteneryzację aplikacji, pozwalając na uruchomienie jej na dowolnym środowisku, z odpowiednią, wystarczającą konfiguracją. Dzięki niemu oprogramowanie może być uruchomione na każdym komputerze [29].

AWS EC2

AWS EC2 jest usługą chmurową stworzoną przez Amazon, umożliwiającą wykorzystanie instancji wirtualnej maszyny. Została użyta do wdrożenia aplikacji i udostępnienia jej dla zewnętrznych użytkowników [30].

AWS S3

AWS S3 jest usługą dostarczoną przez Amazon, stworzoną do przechowywania danych w formie obiektów. Mogą być to pliki, zdjęcia, filmy, itd. Zapisane tam obiekty mogą być organizowane i dzielone na koszyki (ang. bucket) [31].

3.2. Narzędzia wykorzystane do testów bezpieczeństwa

Ettercap

Ettercap jest dedykowanym narzędziem do przeprowadzania ataków człowiek w środku. Oferuje usługi do zatruwania połączeń, filtrowania przechwyconych pakietów, a także wiele innych możliwości do realizacji ataku [14].

Wireshark

Wireshark jest oprogramowaniem, które umożliwia śledzenie, przechwytywanie i nagrywanie dużej ilości pakietów, transmitowanych z użyciem wielu, różnorodnych protokołów komunikacyjnych. W połączeniu z innymi narzędziami jak np. Ettercap, pozwala na wykonywanie ataków człowiek w środku [15].

Nmap

Nmap jest otwartym narzędziem do przeprowadzania audytów i przeglądu bezpieczeństwa sieci. Pozwala na szybkie i wszechstronne skanowanie sieci opartych na

protokole TCP/IP. Do wykrywania dostępnych adresów, usług, wraz z ich systemami operacyjnymi, stosowanych filtrów, a także wielu innych informacji sieciowych, wykorzystuje niskopoziomowe pakiety IP [13].

Jwt io

Jwt to jest aplikacją internetową, służącą do operacji na tokenach JWT. Z jej pomocą w prosty sposób możliwe jest odkodowanie, a także modyfikacja tokenów JWT, wykorzystywanych do uwierzytelniania i autoryzacji w aplikacjach.

Burp Suite

Burp Suite jest zaawansowanym oprogramowaniem, służącym do przeprowadzania testów aplikacji internetowych. Pozwala na przechwytywanie, modyfikowanie oraz wysyłanie żądań z oraz do aplikacji internetowych, za pomocą funkcjonalności proxy. Posiada rozbudowaną gamę rozwiązań ułatwiających przeprowadzanie testów penetracyjnych takich jak skaner, służący do wykrywania podatności wstrzyknięcie (ang. injection), czy Repeater – umożliwiający wysyłanie powtarzających się automatycznie modyfikowanych zapytań [17].

3.3. Analiza kodu pod kątem bezpieczeństwa

W poniższym rozdziale analizie został poddany kod utworzonej aplikacji - 4referee. Zamieszczone zostały fragmenty kodu wskazujące na potencjalnie występujące podatność wraz z jej opisem oraz sugestią naprawy.

3.3.1. Ujawnione dane uwierzytelniające

Niebezpieczny fragment kodu

```
private AWSCredentials credentials() {
    return new BasicAWSCredentials("accessKey", "secretKey");
}
```

Listing 3.1. Dane uwierzytelniające zawarte w kodzie.

Opis błędu

We fragmencie kodu zamieszczony na listingu 3.1, można zauważyć udostępnione - klucz dostępu (ang. access key) i klucz tajny (ang. secret key) - oba z nich zostały zmienione na potrzeby pokazania błędu. Jako, że aplikacja korzysta z chmury AWS, klucze te są niezbędne do uzyskania dostępu do wykorzystanych usług. Dane te mogą pozwalać na dostęp do wszystkich zasobów chmurowych, jak również informacji o płatnościach. Takie udostępnienie kluczy powoduje, że każda osoba mająca dostęp do repozytorium może uzyskać dostęp do usług chmurowych.

Potencjalne rozwiązanie

Jedynymi osobami mającymi dostęp do kluczy powinien być administrator. Przechowywanie takich danych w kodzie, wiąże się z ryzykiem ujawnienie ich osobom niepożądanym. Jednym z rozwiązań jest przechowywanie nie tylko tych, ale wszystkich danych uwierzytelniających w serwisach do tego przeznaczonych np. AWS Secret Manager.

3.3.2. Błędna konfiguracja zabezpieczeń

Niebezpieczny fragment kodu

```
@Bean
SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
   return http
         .headers()
         .and()
         .csrf().disable()
         .cors()
         .and()
         .authorizeRequests()
         .antMatchers("/api/register")
         .permitAll()
         .antMatchers("/h2-console/**")
         .permitAll()
```

Listing 3.2 Implementacja konfiguracji łańcucha zabezpieczeń.

Opis błędu

Na konfiguracji łańcucha zabezpieczeń zamieszczonej na listingu 3.2, można zauważyć dwa błędy. Pierwszym z nich jest wyłączenie ochrony przed atakami CSRF – domyślnie włączonymi przez bibliotekę Spring Security, umożliwiając tym samym ataki CSRF. Drugim błędem, który można tu dostrzec, jest udostępnienie dostępu do bazy h2. Jest ona wykorzystywana podczas implementacji aplikacji jako lekka, wbudowana i łatwa w konfiguracji baza danych, aby opóźnić wybór ostatecznego rozwiązania. Zezwolenie na dostęp na podany adres wszystkim użytkownikom, może powodować podgląd bazy danych przez nieuprawnione osoby.

Potencjalne rozwiązanie

Ochronę przed atakami CSRF można zapewnić na dwa sposoby:

- Włączyć domyślną ochronę zaimplementowaną przez Spring Security.
- Skonfigurować politykę sesji jako bezstanową (ang. stateless), która będzie wskazywała, że aplikacja nie tworzy sesji http.

Drugie z tych rozwiązań wydaje się być wystarczające, jako że aplikacja korzysta z uwierzytelnienia przy użyciu JWT Token, nie wykorzystując mechanizmu sesji HTTP. W przypadku udostępnieniu bazy h2 dla wszystkich żądań, należy niezwłocznie ograniczyć dostęp, tylko dla użytkowników realnie go potrzebujących.

3.3.3. Użycie słabego klucza podpisu dla JWT Token

Niebezpieczny fragment kodu

```
private static final String JWT_SIGNING_KEY = "signing_key";

private static String createToken(Map<String, Object> claims,
UserDetails userDetails) {

   return Jwts
                .builder()
        .setClaims(claims)
        .setSubject(userDetails.getUsername())
        .claim("authorities", userDetails.getAuthorities())
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() +
TimeUnit.MINUTES.toMillis(EXPIRATION_TIME_MINUTES)))
        .signWith(SignatureAlgorithm.HS256, JWT_SIGNING_KEY)
        .compact();

}
```

Listing 3.3. Fragment kodu tworzący JWT Token dla serwera.

Opis błędu

Podobnie jak w przypadku danych uwierzytelniających dla chmury, klucz JWT nie powinien przechowywany być w kodzie, tak jak jest to widoczne na listingu 3.3. Dodatkowo jako kluczowa część JWT Token, jego siła powinna być większa, a sam klucz powinien być co jakiś czas zmieniany. Uzyskanie klucza przez atakującego pozwoli mu na stworzenie własnego, prawidłowego klucza JWT.

Potencjalne rozwiązanie

Klucz JWT powinien być przechowywany poza kodem, dobrym do tego miejscem może być AWS Secret Manager. Pozwoli to powielić rozwiązania związane z danymi uwierzytelniającymi AWS, co ułatwi zarządzanie i rozwój oprogramowania. AWS Secret Manager pozwala również na zastosowanie polityki rotowania klucza, przez co będzie on jeszcze bardziej odporny na potencjalny wyciek.

3.3.4. Brak walidacji przy pobieraniu użytkownika po id

Niebezpieczny fragment kodu

```
@GetMapping(path = "/{userId}")
ResponseEntity<UserReadDto> getUser(@PathVariable("userId") Long userId) {
   return ResponseEntity.ok(userService.getUserById(userId));
}

UserReadDto getUserById(Long id) {
   return userRepository.findById(id)
         .map(UserReadDto::new)
         .orElseThrow(() -> new NoSuchElementException(String.format("User with given id: %d not
exist", id)));
```

Listing 3.4. Fragment kodu, przetwarzający żądanie do serwera w celu pobrania danych użytkownika po id. Metoda getUser znajduje się w kontrolerze, a metoda getUserById jest metodą serwisu.

Opis błędu

Użytkownik, który poprawnie przeszedł proces uwierzytelniania, może uzyskać dostęp do danych każdego użytkownika. Wystarczy, że po poprawnym logowaniu użyje swojego JWT Token i wyśle żądanie z dowolną wartością „userId", która istnieje w bazie danych. Tak jak zostało to zaprezentowane na listingu 3.4, kontroler bez żadnej walidacji wywoła metodę serwisu, która następnie wywoła metodę repozytorium i zwróci nieautoryzowanemu atakującemu dane innego użytkownika.

Potencjalne rozwiązanie

W Spring Security do uwierzytelniania przy użyciu JWT Token, wykorzystywane są nazwa użytkownika i hasło, gdzie nazwa użytkownika musi być zaimplementowana jako unikalna, w celu poprawnego działania uwierzytelnienia. Pozwolenie na posiadanie na nieunikalnej nazwy użytkownika, prowadziłoby do błędów. Dane o zalogowanym użytkowniku przechowywane są w klasie biblioteki - SecurityContextHolder, z którego można pobrać informację o aktualnie zalogowanym użytkowniku. W ten sposób, istnieje możliwość sprawdzenia, czy osoba wysyłająca żądanie z podanym identyfikatorem jest osobą realnie zalogowaną do aplikacji, poprzez porównanie jej nazwy użytkownika.

Innym rozwiązaniem mogłoby być niewykorzystywanie żadnych parametrów przesyłanych od użytkownika, a jedynie pobieranie aktualnie zalogowanego użytkownika przy użyciu unikalnej jego nazwy. Konkretna implementacja zależy od wielu czynników, natomiast w tej formie, powyższa metoda umożliwia nieautoryzowane uzyskanie dostępu do danych innych użytkowników.

3.3.5. Niewystarczające wymagania złożoności hasła

Niebezpieczny fragment kodu

```
@Pattern(regexp = "^(?=.*\\d).{8,30}$",
        message = "Password must contain at least one number")

@Size(min = 8,
     max = 30,
     message = "Password should be between 8 and 30 digits")

private String password;
```

Listing 3.5. Fragment kodu sprawdzający poprawność przesłanego hasła.

Opis błędu

Hasło powinno być realnym zabezpieczeniem konta przed włamaniem. W związku z tym, powinno się żądać od użytkownika takiej jakości haseł, żeby było to dla niego bezpieczne. Na listingu 3.5 zaprezentowane zostały wymagania dotyczące hasła w aplikacji - o długości ośmiu znaków, zawierające jedną cyfrę. Hasło takie może być bardzo szybko złamane i powszechnie uznawane jest za nie gwarantujące odpowiedniego poziomu bezpieczeństwa.

Potencjalne rozwiązanie

Rozwiązaniem tego problemu może być zwiększenie wymagań co do hasła np. do dziesięć znaków w tym: numeru, dużej i małej litery oraz symbolu. Czas łamania takiego hasła zgodnie z rysunkiem 2.2 wynosi około dwa tygodnie. Dla zmotywowanego cyberprzestępcy nie jest to dużo, natomiast taki atak przy odpowiednim monitoringu, łatwo wykryć na poziomie aplikacji. Należy również znaleźć balans pomiędzy komfortem użytkownika, a realnym zagrożeniem. Z tej perspektywy wymaganie hasła o długości np. 20 znaków, czy włączenie logowania dwuskładnikowego, mogłoby być dla przeciętnej osoby korzystającej z oprogramowania zbyt dużym narzutem. Aplikacja do przetwarzania plików multimedialnych na wczesnym etapie, nie jest potencjalnym źródłem ważnych danych, stąd też zestawiając potencjalne ryzyko, z prawdopodobieństwem ataku i jego szkodliwością, należy przyjąć, że dziesięcio-znakowe, złożone hasło, będzie wystarczającym zabezpieczeniem.

3.3.6. Brak walidacji video

Niebezpieczny fragment kodu

```
@PostMapping(consumes = {MediaType.MULTIPART_FORM_DATA_VALUE})

ResponseEntity<VideoReadDto> addVideo(@RequestParam String videoDataWriteDto,
                        @RequestPart MultipartFile video,
                        @RequestPart MultipartFile videoThumbnail) {

   ObjectMapper objectMapper = new ObjectMapper();
   try {

      VideoDataWriteDto videoData =        objectMapper.readValue(videoDataWriteDto,
VideoDataWriteDto.class);
      VideoReadDto videoReadDto = this.videoService.addVideo(videoData, video, videoThumbnail);

      return ResponseEntity.status(HttpStatus.CREATED).body(videoReadDto);
```

Listing 3.6. Fragment kodu obsługujący żądanie przesłania plików multimedialnych do aplikacji.

Opis błędu

W powyższym fragmencie kodu (listing 3.6) można zauważyć, że w otrzymanym od użytkownika żądaniu video oraz jego miniatura nie są w żaden sposób sprawdzane. Pomimo że frontend aplikacji akceptuje tylko pliki multimedialne, a obraz jest generowany automatycznie, to w dalszym ciągu oprogramowanie jest podatne na wszelkiego rodzaju żądania pochodzące z narzędzi, takich jak Postman lub Curl, a także te modyfikowane przy użyciu oprogramowania Burp. Pozwala to na wysyłanie do aplikacji backend żądań zawierających pliki tekstowe, pliki wykonywalne oraz wszelkiego rodzaju złośliwe oprogramowanie.

Potencjalne rozwiązanie

Przychodzące pliki należy poddać dokładnej kontroli, nie można w tej materii polegać na widoku aplikacji. Koniecznym jest sprawdzenie wielkości wysyłanego pliku, a także jego rozszerzenia. Jeśli oczekiwany jest plik video mogą to być MP4, MOV, AVI itd.

3.3.7. Przyjmowanie danych video jako typ string

Niebezpieczny fragment kodu

```
ResponseEntity<VideoReadDto> addVideo(@RequestParam String videoDataWriteDto,
                        @RequestPart MultipartFile video,
                        @RequestPart MultipartFile videoThumbnail) {

   ObjectMapper objectMapper = new ObjectMapper();
   try {

      VideoDataWriteDto videoData = objectMapper.readValue(videoDataWriteDto,
```

```
VideoDataWriteDto.class);
        VideoReadDto videoReadDto = this.videoService.addVideo(videoData, video, videoThumbnail);
```
Listing 3.7. Fragment kodu obsługujący żądanie przesłania plików multimedialnych do aplikacji.

Opis błędu

Jest to drugi błąd we fragmencie kodu obsługującym żądanie przesłania plików multimedialnych przedstawionym na listingu 3.7. W tym wypadku jest to przyjmowanie danych z formularza od użytkownika jako typ string, a następnie wykorzystanie klasy ObjectMapper do przetworzenia ich na Obiekt, bez wcześniejszej walidacji. Implementacja takiego sposobu przetwarzania danych, powoduje rezygnację z użycia wbudowanych mechanizmów biblioteki Spring do mapowania obiektów. Mechanizmy te zawierają w sobie wiele funkcji ochrony przed atakami opartymi na wstrzyknięciach takimi jak, XSS, SQL Injection itd. Dodatkowo przetwarzanie żądań przy pomocy ObjectMapper, wyłącza możliwość automatycznego sprawdzenia opartego na adnotacjach, działającej już na etapie wpłynięcia żądania i wymuszając na programiście dokładną implementację mechanizmów walidacji. Procesowanie danych pochodzących od użytkownika w ten sposób powoduje, że aplikacja jest narażona na ataki z użyciem złośliwego kodu zawartego w wartości typu string, wszelkie wstrzyknięcia, czy niespodziewane struktury danych.

Potencjalne rozwiązanie

Najlepszym rozwiązaniem jest wykorzystanie wbudowanych mechanizmów biblioteki Spring do automatycznego mapowania przesyłanego obiektu w ciele żądania w formacie JSON (JavaScript Object Notation) na obiekt. Rozwiązanie to nie jest jednak możliwe przy wykorzystaniu @RequestPart do przesyłania plików. Potencjalne rozwiązania są więc dwa:

- Użycie dwóch węzłów końcowych (ang. endpoint) do obsługi żądań. Jeden do przyjmowania detali video jako ciało żądania, drugi do obsługi plików multimedialnych. Zapewni to sprawdzenie wartości przychodzących z formularza. Wadą tego podejścia jest skomplikowana synchronizacja zapisu żądań, pochodzących z dwóch węzłów końcowych dla jednej encji. Może to również prowadzić do błędów, w którym jedno z żądań nie zostanie wykonane, a dane z drugiego zostaną zapisane w bazie danych. Wtedy informacje o obiekcie będą niekompletne, co będzie mogło prowadzić do niespójności danych np. istnienia video, bez przypisanego do niego linku.

- Drugim proponowanym rozwiązaniem jest napisanie mechanizmu walidacji, który będzie sprawdzał przychodzące dane w formacie typu string, pod kątem mogących wystąpić zagrożeń. To rozwiązanie pozostawi logikę zapisu video niezmienioną, ale wprowadzi bardzo duży narzut pracy i odpowiedzialności na programistę, muszącego poprawnie przetwarzać przychodzące żądanie, pod kątem potencjalnych wstrzyknięć czy ataków XSS.

3.3.8. Łączenie dwóch wartości typu string

Niebezpieczny fragment kodu

```
    Video video = videoToAdd.toVideo(user);
    video.setTitle(generateTitle(savedDecision));
    video.setVideoUrl(uploadedVideoUrl);
    video.setThumbnailUrl(uploadedVideoThumbnailUrl);
    return new VideoReadDto(videoRepository.save(video));
}

private String generateTitle(Decision decision) {

return String.format("%s_%s", decision.getMatchMinute(), decision.getTitle);
}
```

Listing 3.8. Fragment kodu przedstawiający złączenie wartości typu tekstowego przed zapisaniem ich do bazy danych.

Opis błędu

Błąd przedstawiony na listingu 3.8, pozwala na połączenie dwóch zmiennych typu string, bez ich uprzedniej walidacji, a następne zapisanie ich do bazy danych. Może to powodować wykonywanie wstrzyknięć podatnych danych, w celu wykonania wstrzyknięcia kodu SQL i uzyskania dostępów do informacji w bazie danych.

Potencjalne rozwiązanie

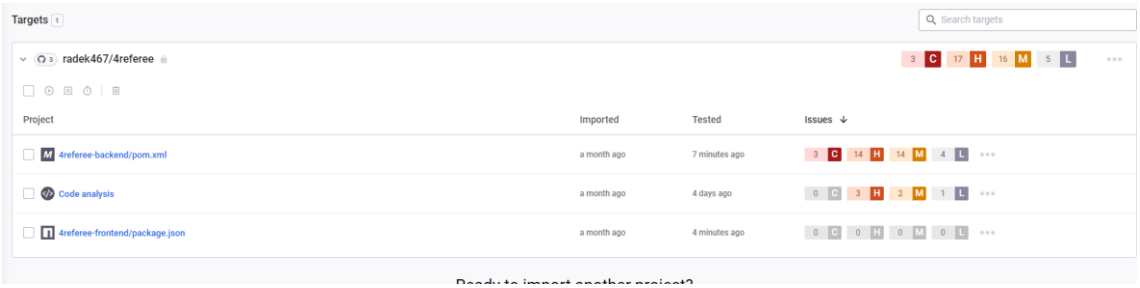
Dane pochodzące od użytkownika należy dokładnie sprawdzać. Biblioteka Spring Data posiada wbudowane walidacje, ale tylko dla jednej zmiennej. Łącząc ze sobą kilka zmiennych tekstowych nie można mieć pewności, że dane zostały poprawnie zweryfikowane. Należy uniknąć łączenia zmiennych ze sobą albo weryfikować je w taki sposób, aby mieć pewność, że atak polegający na wstrzyknięciu nie będzie możliwy.

3.4. Statyczna analiza podatności narzędziem Snyk

Prezentacja wykrytych podatności

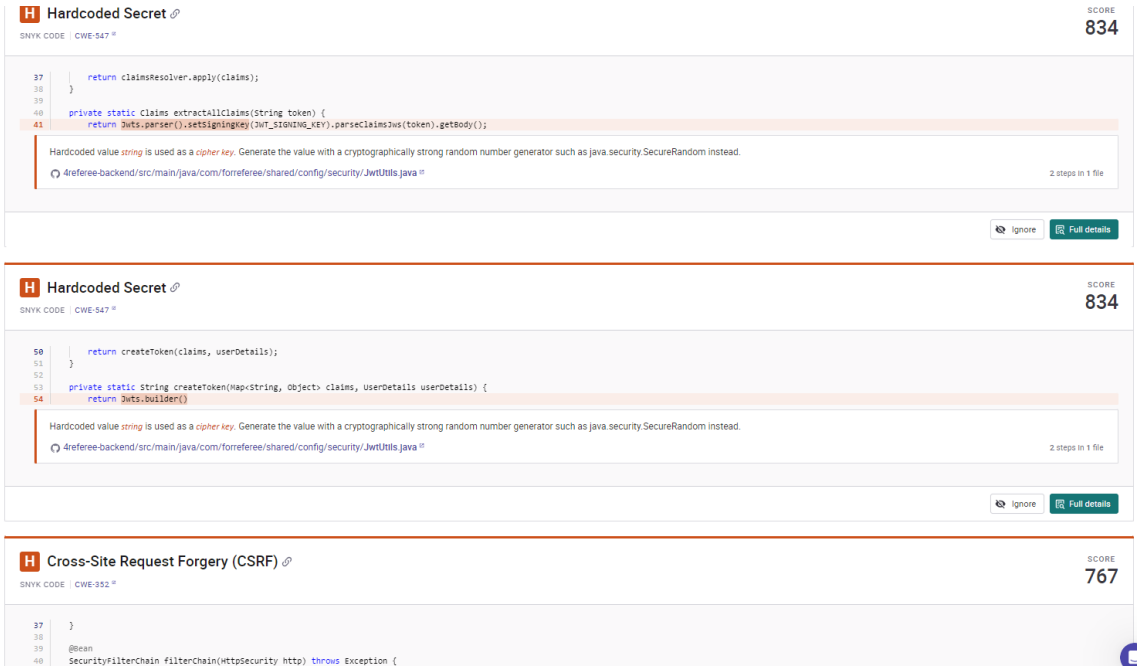
Analiza podatności narzędziem Snyk, pozwoliła wykryć 41 podatności znajdujących się w aplikacji. Trzy z nich określono jako krytyczne, siedemnaście oceniono jako wysokie, szesnaście jako średnie i tylko pięć jako niskie. Sześć z nich

dotyczą stworzonego kodu, pozostałe trzydzieści pięć zostało dostarczone do aplikacji z wykorzystanych bibliotek. Podsumowanie podatności widoczne są na rysunku 3.11.



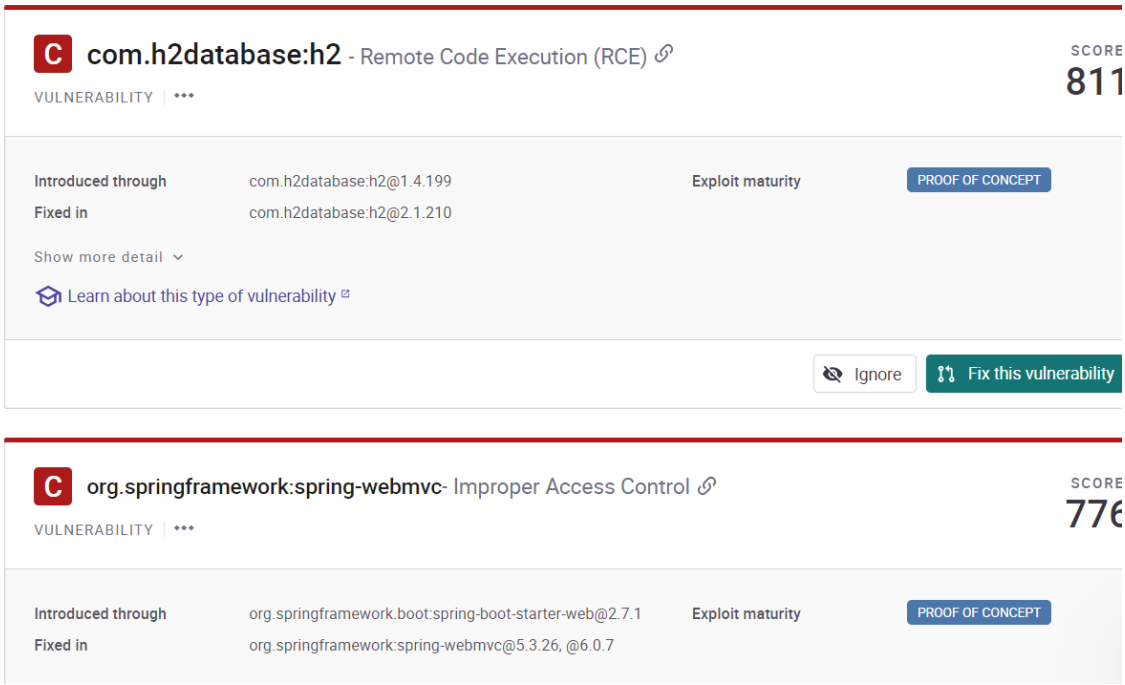
Rysunek 3.11. Podsumowanie podatności wykrytych przez narzędzie Snyk.
Źródło: opracowanie własne

Podatności wykryte na podstawie implementacji dotyczą głównie ujawnionych w kodzie kluczy, a także wyłączonej ochrony przed atakami csrf (rysunek 3.12).



Rysunek 3.12. Podatności wykryte w kodzie aplikacji.
Źródło: opracowanie własne

Zaskakującym rezultatem jest ilość podatności odkrytych w zewnętrznych bibliotekach, których wykrycie bez narzędzia do statycznej analizy kodu nie byłoby możliwe. Część z wykrytych podatności widoczne są na rysunku 3.13.



Rysunek 3.13. Wykryte podatności w wykorzystanych bibliotekach.
Źródło: opracowanie własne

Snyk dostarczył informacje o podatnościach, udostępnił dokładne ich opisy, a także sugerowane drogi rozwiązania. Informuje również, czy istnieje powszechnie dostępny atak (ang. exploit), wykorzystujący podatność.

W poniższej tabeli (tabela 3.1) zostały przedstawione wykryte podatności, ich pochodzenie, oraz ocena dokonana przez narzędzie.

Tabela 3.1. Wykryte podatności

Podatność	**Pochodzenie**	**Ocena**	**Opis podatności**
Remote Code Execution	com.h2database:h2@1.4.199	811	Podatność ta pozwala użytkownikowi na wstrzyknięcie złośliwego kodu do aplikacji poprzez dane wejściowe. Podatność została naprawiona w wersji 2.1.210. W celu pozbycia się niebezpieczeństwa, należy zaktualizować oprogramowanie do nowszej wersji.
Denial of Service	org.springframework.boot:spring-boot-starter-web@2.7.1	804	Podatność ta pozwala atakującemu na blokadę usług, przy użyciu wyrażenia regex. Przy pomocy przechodnich zależności podatna biblioteka org.apache.tomcat, została dodana do

			aplikacji przez bibliotekę spring-booot-statet-web. W celu pozbycia się podatności należy zaktualizować oprogramowanie do nowszej wersji.
Improper Access Control	org.springframework.boot:spring-boot-starter-web@2.7.1	776	Podatność ta, może powodować możliwość odczytania wrażliwych danych przez użytkowników nieposiadających właściwych uprawnień. Przy pomocy przechodnich zależności podatna biblioteka spring-webmvc, została dodana do aplikacji przez bibliotekę spring-booot-stater-web. W celu pozbycia się podatności należy zaktualizować narzędzie do nowszej wersji.
Access Control Bypass	org.springframework.boot:spring-boot-starter-security@2.7.1	776	Podatność ta pozwala na uzyskanie nieautoryzowanego dostępu do zasobów. Została wprowadzona przez bibliotekę spring-security. W celu pozbycia się podatności należy zaktualizować bibliotekę do nowszej wersji.
Remote Code Execution	com.h2database:h2@1.4.199	726	Podatność ta pozwala użytkownikowi na wstrzyknięcie złośliwego kodu do aplikacji poprzez dane wejściowe zapisywane w bazie danych. Podatność została naprawiona w wersji 2.1.206. W celu pozbycia się podatności należy zaktualizować wersję oprogramowania.
XML External Entity Injection	com.h2database:h2@1.4.199	726	Podatność ta została wprowadzona przez użycie biblioteki h2 w wersji 1.4.199 i pozwala na atak typu XXE (XML External Entity Attack). W celu wyeliminowania podatności należy zaktualizować wersję biblioteki h2 do wersji 2.0.202.

Arbitary Code Execution	org.springframework.boot:spring-boot-starter-validation@2.7.1 – snake.yaml	651	Podatność ta została wprowadzona do kodu z wykorzystaniem zależności przechodnich przez bibliotekę Spring Boot, która korzysta z biblioteki snake.yaml. Pozwala na wykonanie dowolnego kodu spowodowanego niewłaściwym sprawdzeniem danych przychodzących. W celu naprawy podatności należy zaktualizować oprogramowanie do nowszej wersji.
Improper Input Validation	org.springframework.boot:spring-boot-starter-web@2.7.1	644	Podatność ta może powodować możliwość odczytania wrażliwych danych przez użytkowników nieposiadających właściwych uprawnień. W celu pozbycia się podatności należy zaktualizować bibliotekę do nowszej wersji.
Denial of Service	com.fasterxml.jackson.datatype:jackson-datatype-jsr310@2.13.4, io.jsonwebtoken:jjwt@0.9.1	616	Wykorzystanie podatności przez atakującego może powodować blokadę usług, z powodu wyczerpania zasobów podczas przetwarzania danych w tablicach wykorzystywanych przez bibliotekę. Problem został naprawiony w wersjach 2.12.7.1 i wyższych.
Denial of Service	com.amazonaws:aws-java-sdk-s3@1.12.561 org.springframework.boot:spring-boot-starter-web@2.7.1	616	Użyte biblioteki podatne są na atak DoS poprzez wykorzystanie tablicy w klasie StdDeserializer. Ta podatność jest możliwa do zrealizowania przez atakującego tylko wtedy, gdy aplikacja korzysta z niedomyślnego rozwiązania „UNWRAP_SINGLE_VALUE_ARRAYS ”. Jako, że aplikacja nie wykorzystuje tego rozwiązania, można podatność zignorować, lub oznaczyć ją jako fałszywie pozytywną.

Improper Input Validation	org.springframework.boot:spring-boot-starter-web@2.7.1 - org.apache.tomcat.embed:tomcat-embed-core	589	Podatność ta do projektu została wprowadzona przez Spring-boot starter web za pomocą zależności przechodnich. Podatnym komponentem jest Tomcat – niewłaściwie weryfikuje on dane wejściowe, w wyniku czego JsonErrorReportValue nie powoduje ucieczki wartości typu. Przy właściwej manipulacji danymi przez użytkownika, błąd może powodować zwrócenie niepożądanych wartości JSON. W celu zlikwidowania podatności, należy zaktualizować do nowszej wersji bibliotekę Spring Boot albo tylko bibliotekę Tomcat'a.
Denial of Service	org.springframework.boot:spring-boot-autoconfigure	589	Podatność ta informuje o możliwej blokadzie usług, wynikającej z jednoczesnego użycia domyślnej biblioteki Spring MVC i odwróconego proxy pamięci podręcznej (ang. cache). Jako że te mechanizmy nie są wykorzystywane w aplikacji, nie jest ona podatna.
Denial of Service	org.yaml:snakeyaml	589	Wykryta podatność przez Snyk informuje o możliwym ataku DoS, wynikającym z braku walidacji głębokości zagnieżdżeń w kolekcjach przekazywanych do aplikacji. Zaktualizowanie wersji biblioteki snake.yaml rozwiązuje ten problem.
Denial of Service	org.springframework.boot:spring-boot-starter-	589	Wykryta podatność przez Snyk informuje o możliwym ataku DoS, wynikającym z braku walidacji głębokości zagnieżdżeń w kolekcjach przekazywanych do aplikacji.

	validation@2.7.1		Zaktualizowanie wersji biblioteki snake.yaml rozwiązuje ten problem.
Authorization Bypass	org.springframework.boot:spring-boot-starter-security@2.7.1	584	Podatność umożliwia obejście autoryzacji. Wprowadzona została przez bibliotekę Spring-Security w wersji 2.7.1. Aplikacja jest narażona na ataki, jeśli przekazuje dalej żądania przychodzące. W tym przypadku tak się nie dzieje, więc alarm można oznaczyć jako fałszywie pozytywny.
Denial of Service	org.springframework.boot:spring-boot-starter-validation@2.7.1 - ch.qos.logback:logback-classic	569	Podatność została wprowadzona za pomocą zależności przechodnich przez bibliotekę Spring Boot Validation, która korzysta z biblioteki ch.qos.logback. Atakujący przy pomocy odpowiednio zatrutych danych może doprowadzić do blokady usług.
Uncontrolled Resource Consumption	org.springframework.boot:spring-boot-starter-validation@2.7.1 - ch.qos.logback	569	Podatność została wprowadzona za pomocą zależności przechodnich przez bibliotekę Spring Boot Validation, która korzysta z biblioteki ch.qos.logback. Problem występuje tylko wtedy, gdy komponent logback-receiver jest włączony i dostępny dla atakującego. W przypadku tej aplikacji tak się nie dzieje, więc alarm można oznaczyć jako fałszywie pozytywny.
Denial of Service	ch.qos.logback:logback-core	569	Podatność została wprowadzona za pomocą zależności przechodnich przez bibliotekę Spring Boot Validation, która korzysta z biblioteki ch.qos.logback. Atakujący przy pomocy odpowiednio zatrutych danych może doprowadzić do blokady usług. Błąd

			można naprawić poprzez zaktualizowanie wersji biblioteki.
Uncontrolled Resource Consumption	ch.qos.logback:logback-core	569	Podatność została wprowadzona za pomocą zależności przechodnich przez bibliotekę Spring Boot Validation, która korzysta z biblioteki ch.qos.logback. Problem występuje tylko wtedy, gdy komponent logback-receiver jest włączony i dostępny dla atakującego. W przypadku tej aplikacji tak się nie dzieje, więc alarm można oznaczyć jako fałszywie pozytywny.
Remote Code Execution	com.h2database:h2@1.4.199	546	Podatność ta jest wprowadzona przez bazę danych h2, która udostępnia konsolę do zarządzania nią. Jednak, aby podatność zaistniała, hasło musi być domyślnie nieustawione, co w przypadku aplikacji nie zostało spełnione. Alarm można oznaczyć jako fałszywie pozytywny.
Access Restriction Bypass	org.apache.tomcat.embed:tomcat-embed-core	539	Podatność została wprowadzona za pomocą zależności przechodnich przez Spring Boot Starter Web. Aby atak mógł być wykonany aplikacja musi korzystać z uwierzytelniania w Spring Security określanego jako „form”, co nie jest wykorzystywane w aplikacji. Alarm można oznaczyć jako fałszywie pozytywny.
Denial of Service	org.apache.tomcat.embed:tomcat-embed-core	539	Podatność wynika z wykorzystywania serwera Tomcat przez bibliotekę Spring Boot Starter Web. Atakujący może doprowadzić do blokady usług wysyłając dużą serię części żądań albo jedno duże żądanie pliku multipart. Podatność można zlikwidować aktualizując wersję biblioteki.

Allocation of Resources Without Limits or Throttling	org.springframework:spring-expression	539	Podatność wynika z wykorzystywanego przez Spring Boot języka SpEL. Atak jest możliwy do wykonania, jeśli atakujący dostarczy bardzo długie wyrażenie wykorzystywane przez bibliotekę, doprowadzając do blokady usług. W celu zlikwidowania podatności należy zaktualizować wersję biblioteki do wersji sugerowanej przez Snyk.
Stack-based Buffer Overflow	org.yaml:snakeyaml	536	Podatność może powodować przeciążenie stosu, a następnie prowadzić do blokady usług. Atakujący w celu zablokowania aplikacji, musi wysłać odpowiednio zmodyfikowany plik YAML. W celu naprawy podatności należy zaktualizować wersję biblioteki org.yaml:snakeyaml.
Session Fixation	org.springframework.boot:spring-boot-starter-security@2.7.1	509	Podatność została wprowadzona przez bibliotekę Spring Security i dotyczy błędów w obsłudze sesji użytkownika po wylogowaniu. Jako że aplikacja korzysta z JWT Token do uwierzytelniania, alarm można oznaczyć jako fałszywie pozytywny.
Session Fixation	org.springframework.boot:spring-boot-starter-security@2.7.1	509	Podatność została wprowadzona przez bibliotekę Spring Security i dotyczy błędów w obsłudze sesji użytkownika po wylogowaniu. Jako że aplikacja korzysta z JWT Token do uwierzytelniania, alarm można oznaczyć jako fałszywie pozytywny.
Stack-based Buffer Overflow	org.springframework.boot:spring-boot-	506	Podatność może powodować przeciążenie stosu, a następnie prowadzić do blokady usług. Atakujący w celu zablokowania

	starter-validation@2.7.1		aplikacji, musi wysłać odpowiednio zmodyfikowany plik YAML. W celu naprawy podatności należy zaktualizować wersję biblioteki org.yaml:snakeyaml.
Improper Input Validation	org.springframework.boot:spring-boot-starter-web@2.7.1 - org.apache.tomcat.embed:tomcat-embed-core	479	Podatność wynika z wykorzystywania serwera Tomcat przez bibliotekę Spring Boot Starter Web. Problem dotyczy błędnej obsługi przychodzących w żądaniu nagłówków HTTP. Atakujący może zmusić serwer, aby potraktował jedno żądanie jako kilka i dzięki temu przemycić zapytania do serwera.
Incomplete Cleanup	org.apache.tomcat.embed:tomcat-embed-core - org.apache.tomcat.embed:tomcat-embed-core	479	Podatność wynika z wykorzystywania serwera Tomcat przez bibliotekę Spring Boot Starter Web. Serwer czyści niektóre dane żądania przed odpowiedzią, natomiast błąd w wewnętrznej implementacji może spowodować pominięcie niektórych danych, a tym samym osoba atakująca może uzyskać nieautoryzowany dostęp do zasobów. W celu naprawy podatności należy zaktualizować wersję biblioteki.
Unprotected Transport of Credentials	org.springframework.boot:spring-boot-starter-web@2.7.1 - org.apache.tomcat.embed:tomcat-embed-core	479	Podatność wynika z wykorzystywania serwera Tomcat przez bibliotekę Spring Boot Starter Web. Dotyczy przesyłania danych uwierzytelniających i wykorzystywania sesji. Jako że aplikacja nie korzysta z tego typu mechanizmów, błąd można oznaczyć jako fałszywie pozytywny.

Allocation of Resources Without Limits or Throttling	org.springframework.boot:spring-boot-starter-security@2.7.1 and org.springframework.boot:spring-boot-starter-web@2.7.1	479	Podatność została wprowadzona przez podatne biblioteki Spring. Dotyczy alokacji zasobów bez ograniczeń albo ograniczenia przepustowości serwera, poprzez odpowiednio przygotowane wyrażenie języka Spring SpEL. W celu naprawy podatności należy zaktualizować oprogramowanie do nowszej wersji.
HTTP Request Smuggling	org.springframework.boot:spring-boot-starter-web@2.7.1 - org.apache.tomcat.embed:tomcat-embed-core	399	Podatność wynika z wykorzystywania serwera Tomcat przez bibliotekę Spring Boot Starter Web. Pozwala na przemycanie zapytań HTTP, poprzez żądania zawierające niewłaściwą wartość nagłówka Contnent-Length, który nie zostanie poprawnie odrzucony. Podatność ta będzie wyeliminowana po zaktualizowaniu wersji biblioteki.
Stack-based Buffer Overflow	org.springframework.boot:spring-boot-starter-validation@2.7.1	399	Podatność może powodować przeciążenie stosu, a następnie prowadzić do blokady usług. Atakujący wysyłając odpowiednio przygotowany plik YAML, może spowodować błąd w klasie BaseConstructor i zablokować serwer. Podatność ta może zostać wyeliminowana poprzez zaktualizowanie biblioteki do nowszej wersji.
Stack-based Buffer Overflow	org.yaml:snakeyaml	399	Podatność może powodować przeciążenie stosu, a następnie prowadzić do blokady usług. Atakujący wysyłając odpowiednio zmodyfikowane dane, może w przypadku

			braku odpowiedniej walidacji wielkości przychodzących danych, doprowadzić do zablokowania systemu. Podatność ta może zostać wyeliminowana poprzez zaktualizowanie biblioteki do nowszej wersji.
Information Exposure	com.h2database:h2@1.4.199	372	Podatność ta może zostać wykorzystana, gdy konsola bazy danych h2, została uruchomiona za pośrednictwem interfejsu CLI z wykorzystaniem argumentu - webAdminPassword. Jako że taka sytuacja nie ma miejsca, należy podatność oznaczyć jako fałszywie pozytywną.

Źródło: opracowanie na podstawie [16].

Rekomendowane działania

Po dokładnej analizie wszystkich trzydziestu sześciu ujawnionych przez Snyk podatności, jedenaście z nich można oznaczyć jako fałszywie pozytywne. Pozostałe dwadzieścia pięć może być rozwiązane poprzez aktualizację do nowszej wersji wykorzystywanych bibliotek. Jako że często mogą to być czasochłonne działania, w zależności od polityki stosowanej w aplikacji wymagana może być naprawa tylko części z istniejących podatności. Na pewno jest to konieczne dla zagrożeń krytycznych oraz wysokich, z najwyższymi wynikami.

3.5. Realizacja ataku blokada usług

Opis podatności

Aplikacja posiada bardzo wysoki limit na wielkość przesyłanych plików - jest to 100 MB dla video oraz 10MB dla obrazu, co łącznie daje rozmiar ponad 110 MB dla żądania. Przyjęcie go, przetworzenie, a następnie wysłanie do chmury S3, może zająć dużo czasu, w którym aplikacja będzie zalewana kolejnymi żądaniami o dużym rozmiarze, nie nadążając z przetwarzaniem ich.

Wykorzystanie podatności

Do przeprowadzenia ataku stworzony został poniższy skrypt bash (listing 3.9), który przy pomocy narzędzia curl, wysyła czterdzieści żądań http POST, dodających viedo i obraz do zasobów użytkownika i wysyłających je do Amazon S3.

```
#!/bin/bash

for i in {1..40}; do
 echo "Sending request $i"
 curl --location 'http://localhost:8080/api/videos' --header 'Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJVc2VyIiwiZXhwIjoxNzAyMTQ4MDU5LCJpYXQiOjE3MDIxND
A4NTksImF1dGhvcml0aWVzIjpbeyJhdXRob3JpdHkiOiJST0xFX1VTRVIifV19.xMFwGpiIJR-
Aem9k2E9Havq_huhyifHAumUEaMSKTYc' --form 'videoThumbnail=@"sample.jpg"' --form
'video=@"bigtest.mp4"' --form
'videoDataWriteDto="{\"description\":\"sa\",\"match\":\"as\",\"decision\":{\"matchMinute\":\"1\",\"catego
ries\":[\"C\"],\"teamSanction\":\"GOAL_KICK\",\"individualSanctions\":[\"YELLOW_CARD\"]}}"' &
done
```

Listing 3.9. Fragment skryptu bash, służący do przeprowadzenia ataku DoS.

Znak & w skrypcie pozwala na pominięcie oczekiwania na odpowiedź. Co istotne, żądanie to jest możliwe do wykonania tylko dla zalogowanego użytkownika. Należało więc pobrać z autoryzowanego nagłówka żądania JWT Token i umieścić go w ciele polecenia curl. Skrypt może być wykonany wiele razy, zwiększając ilość wysłanych żądań.

Do wygenerowania dużego pliku o wymaganym rozmiarze została wykorzystana komenda head (listing 3.10):

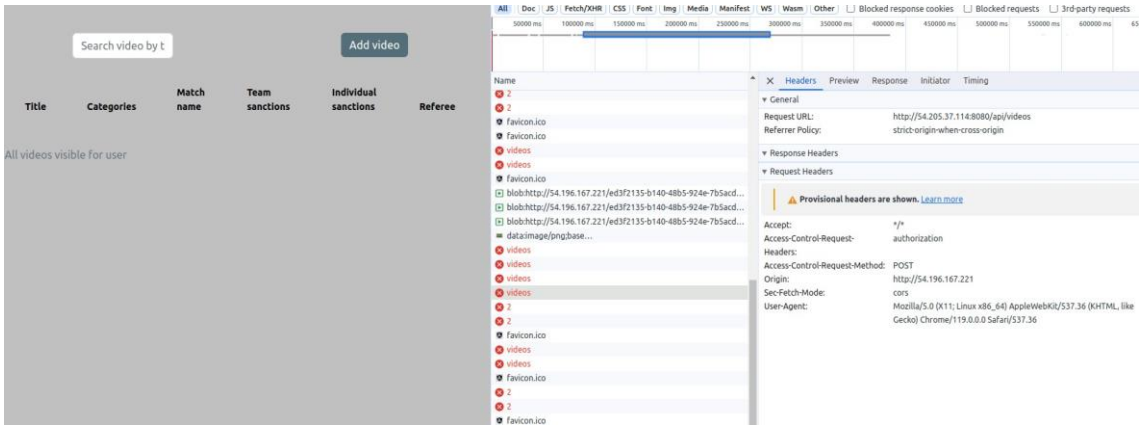
```
head -c 102760448 </dev/urandom >bigtest.mp4
```

Listing 3.10. Polecenie bash tworzący plik o określonym rozmiarze.

Komenda head tworzy plik mp4 o rozmiarze 98 MB. Ta sama procedura została wykorzystana do utworzenia obrazu o wielkości 10 MB.

Efekt przeprowadzonego ataku

Efektem ataku było całkowite zawieszenie systemu po stronie serwerowej. Po ataku, jakiekolwiek próby dodawania video, wyświetlania danych użytkownika, pozostawały bez odpowiedzi ze strony serwera. Każde żądanie po ok. 180 sekundach, zwracało błąd i brak odpowiedzi od serwera, co zostało zaprezentowane na rysunku 3.14. W rezultacie czego korzystanie z aplikacji było niemożliwe.



Rysunek 3.14. Zawieszenie działania aplikacji w wyniku ataku Denial of Service.
Źródło: opracowanie własne

Będąc zalogowanym do konsoli aplikacji w AWS EC2, możliwe było do zauważenia, że aplikacja nie zamieszcza żadnych logów z żądań, co widoczne jest na rysunku 3.15. Jest to bardzo zła praktyka, ponieważ w tym przypadku utrudniało to zlokalizowanie użytkownika przeprowadzającego atak. Nie został zaimplementowany również monitoring sieciowy, który przy nadmiernym ruchu z jednego adresu IP byłby w stanie to dostrzec i taką aktywność zablokować.

Rysunek 3.15. Brak logów w aplikacji z informacjami o otrzymywanych żądaniach.
Źródło: opracowanie własne

Rekomendowane rozwiązanie

Sposobów ochrony przed atakami DoS, jest wiele. W pierwszym kroku należy zweryfikować czy rozmiar pliku 100MB jest realnie potrzebny użytkownikom. Wydaje się, że na przesyłanie krótkich materiałów video jest to zdecydowanie za dużo. Drugą rzeczą konieczną do implementacji jest poprawa logowania zdarzeń w aplikacji. W obecnej wersji oprogramowania, nie są logowane żadne informacje o żądaniach, co uniemożliwia jakiekolwiek działania związane z obroną przed atakiem. Trzecim z rozwiązań jest implementacja monitoringu sieciowego, alarmującego o niestandardowym ruchu, a następnie ewentualne blokowanie go. Wszystkie te akcje, współdziałając ze sobą, nie dadzą gwarancji na całkowite pozbycie się ryzyka ataku DoS, ale powinny pozwolić na zminimalizowanie jego skutków.

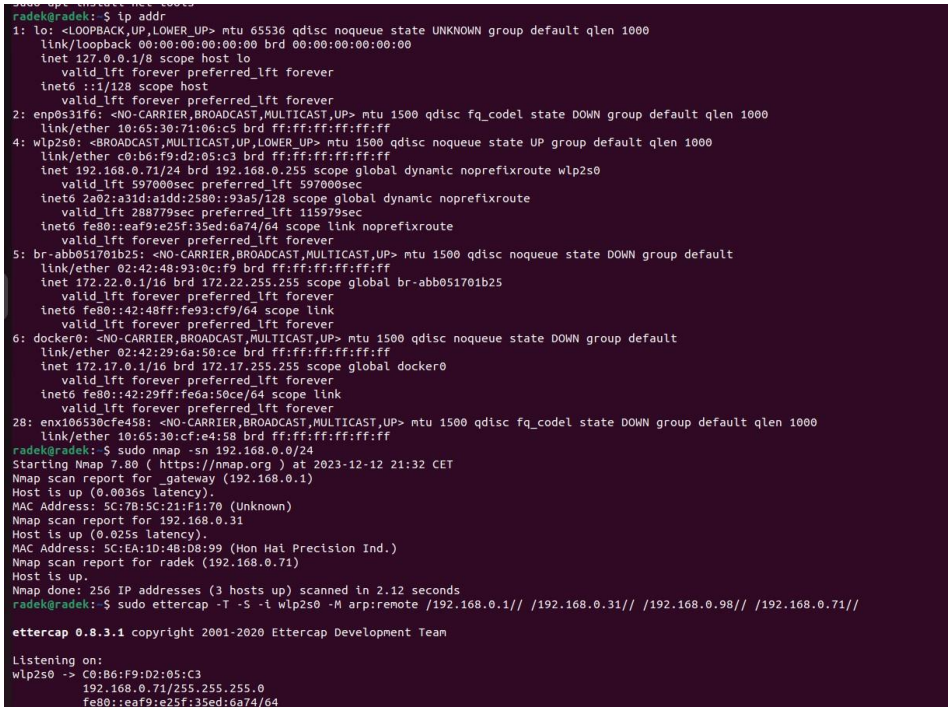
3.6. Realizacja ataku Man in The Middle

Opis podatności

Aplikacja, do komunikacji z użytkownikami wykorzystuje protokół HTTP, nie szyfrując ruchu, tym samym narażając użytkowników na stanie się ofiarami ataku człowiek w środku.

Wykorzystanie podatności

W celu ujawnienia podatności, należało zainfekować urządzenia w routerze, a następnie nasłuchiwać nadchodzącego ruchu HTTP. Do znalezienia wszystkich adresów IP w sieci, wykorzystano narzędzie nmap, które umożliwiło wykonanie skanowania ping (ang. Ping Scan), przy użyciu polecenia -sn, wysyłając żądania ICMP Echo i odkrywając wszystkie aktywne hosty. Następnie wykorzystując narzędzie ettercap dokonano zatrucia tablic arp wszystkich urządzeń w sieci lokalnej. Proces wykonania ataku został przedstawiony na rysunku 3.16.

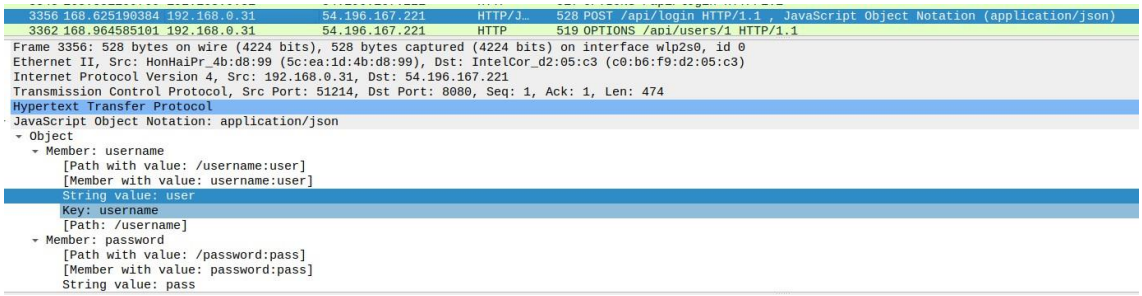


Rysunek 3.16. Wykonanie ataku Man in the Middle, skanowanie sieci i zatrucie tablic ARP urządzeń w sieci lokalnej.
Źródło: opracowanie własne

Efekt przeprowadzonego ataku

Po zatruciu urządzeń znajdujących się w sieci lokalnej, wykorzystując aplikację wireshark, służącą do analizy przepływających pakietów, nasłuchiwano żądania HTTP z jednego z podanych adresów, do adresu IP aplikacji. Przyniosło to efekt w postaci

wykrycia próby logowania i przechwycenia nazwy użytkownika oraz hasła, wysyłanych czystym testem za pomocą protokołu HTTP (rysunek 3.17).



Rysunek 3.17. Odkryte hasło podczas ataku Man in the Middle.
Źródło: opracowanie własne

Rekomendowane rozwiązanie

Przed atakiem można zabezpieczyć się zarówno od strony aplikacji – używając do komunikacji protokołu HTTPS, jak również od strony użytkownika – korzystając z VPN. Jednak aplikacja nie powinna polegać na użytkownikach w kwestiach bezpieczeństwa, licząc, że to oni będą odciążali twórców aplikacji, korzystając z VPN. Tak szybko jak to możliwe, powinna zostać wprowadzona szyfrowana komunikacja poprzez protokół HTTPS.

3.7. Realizacja ataku siłowego

Opis podatności

Aplikacja nie posiada, żadnego mechanizmu captcha, nie blokuje też możliwości logowania po określonej ilości nieudanych logowań, powodując, że w połączeniu z brakiem monitoringu oraz logowania zdarzeń, staje się bardzo łatwym celem do ataków siłowych.

Wykorzystanie podatności

Początkowym założeniem było dokonanie ataku za pomocą narzędzia hashcat, ale po rekonesansie okazało się, że nie można z jego pomocą wysyłać żądań z danymi do logowania w ciele aplikacji. W związku z tym został stworzony własny skrypt, do wykonywania ataku siłowego, bezpośrednio na aplikację backend, dostosowany do wymagań aplikacji, przedstawiony na listingu 3.11.

```
if [ $# -eq 0 ]; then
    echo "Usage: $0 <file>"
    exit 1
fi
if [ ! -f "$1" ]; then
    echo "Error: File not found!"
    exit 1
fi
```

```
while read -r word; do
   if [ -n "$word" ]; then
      response=$(curl -sS -X POST -H "Content-Type: application/json" -d '{"username": "admin",
"password": """$word"""}' "http://localhost:8080/api/login")
      if [[ "$response"  =~ 'FORBIDDEN' ]]; then
         :
      else
         echo "$response"
         echo "POST request for $word was successful (HTTP 200 OK)"
      fi
   fi
done < "$1"
echo "POST requests completed."
```

Listing 3.11. Skrypt bash wykonujący atak Brute Force.

Skrypt przyjmuje argument z nazwą pliku zawierającego hasła, wymaganego do przeprowadzenia ataku słownikowego. Po rozpoczęciu działania sprawdza, czy argument został podany oraz to, czy podany plik istnieje, a następnie zaczyna odczytywanie słów w pętli while. Jeśli linia nie jest pusta i hasło istnieje, zostaje wysłane żądanie logowania do aplikacji backend. Jako że aplikacja w przypadku błędnego logowania zwraca błąd 403 Forbidden, to po spełnieniu tego warunku wykonywane jest jedynie kolejne żądanie, bez zostawiania żadnej informacji dla atakującego. W przypadku gdy logowanie jest zakończone sukcesem, zostają wyświetlone nazwa użytkownika i hasło, a także odpowiedź serwera.

Do ataku została wykorzystana lista haseł oznaczona w bibliografii numerem [9].

Efekt przeprowadzonego ataku

Atak został przeprowadzony skutecznie, ujawnione zostało hasło użytkownika admin, co widoczne jest na rysunku 3.18.

```
radek@radek:~/Projects/mgr/brute-force$ ./brute.sh xato-net-10-million-passwords-1000000.txt
{"id":3,"username":"Admin","email":"admin@4referee.pl","userRole":"ADMIN"}
POST request for pass was successful (HTTP 200 OK)
```

Rysunek 3.18. Odkryte hasło przy pomocy ataku ataku siłowego.
Źródło: opracowanie własne

Podobnie jak w przypadku ataku DoS, można zauważyć problemy z logowaniem zdarzeń. W logach aplikacji, które są widoczne na rysunku 3.19, zapisywane są tylko procesy uwierzytelniania zakończone sukcesem. Tak ubogie informacje uniemożliwiają zauważenie nieskutecznych prób, co z kolei utrudnia wychwycenie i przerwanie ataku.

```
INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 3 ms
INFO 1 --- [io-8080-exec-10] c.f.user.AuthenticationController        : Successful login with username: admin
```

Rysunek 3.19. Brak informacji w logach o nieudanych logowaniach.
Źródło: opracowanie własne

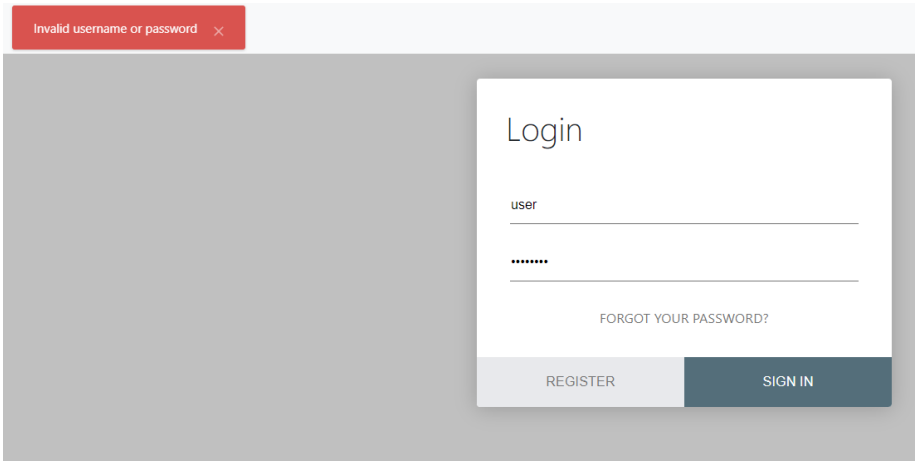
Rekomendowane rozwiązanie

Rozwiązaniem, które należy wprowadzić niezwłocznie, jest ograniczenie ilości nieudanych logowań np. do 5, zamrażając ten proces np. na minutę. Powinno to skutecznie ograniczyć możliwość wykonywania ataków siłowych bezpośrednio na punkt końcowy logowania. Kolejną rzeczą, która nie może pozostać w aktualnej formie jest złożoność haseł, minimalnie 10 różnorodnych znaków, tak jak zostało to opisane w części analizy kodu pod kątem. Przy takiej konfiguracji wykonywanie ataków siłowych będzie miało mniejsze szanse powodzenia. Kolejnym rozwiązaniem nad, którym warto się zastanowić jest wprowadzenie mechanizmu Captcha, która poprawnie zaimplementowany, będzie blokował automatyczne ataki. Istotne zwiększenie bezpieczeństwa zapewniałoby logowanie dwuskładnikowe, natomiast w aplikacji tego typu, o niskim ryzyku ataku i szkodliwości potencjalnego wycieku danych, nie jest to konieczne. Koszt implementacji takiego rozwiązania na wczesnym etapie rozwoju byłby duży, a przy odpowiednim zastosowaniu powyżej opisanych kroków, efekt niewielki.

3.8. Wykryta podatność Security Misconfiguration podczas rejestracji nowego użytkownika

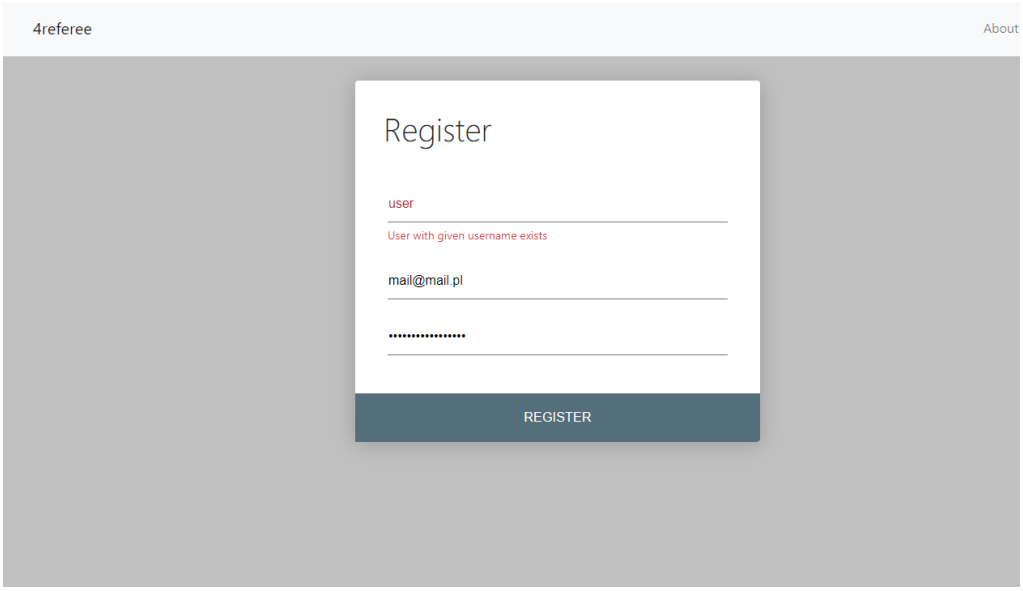
Opis podatności

Podczas logowania do aplikacji, użytkownik, wprowadzając nieprawidłową konfigurację nazwy otrzymuje błąd informujący, że nieprawidłowe są albo nazwa użytkownika albo hasło, co jest właściwą, bezpieczną informacją o błędzie. Działanie to zostało przedstawione na rysunku 3.20.



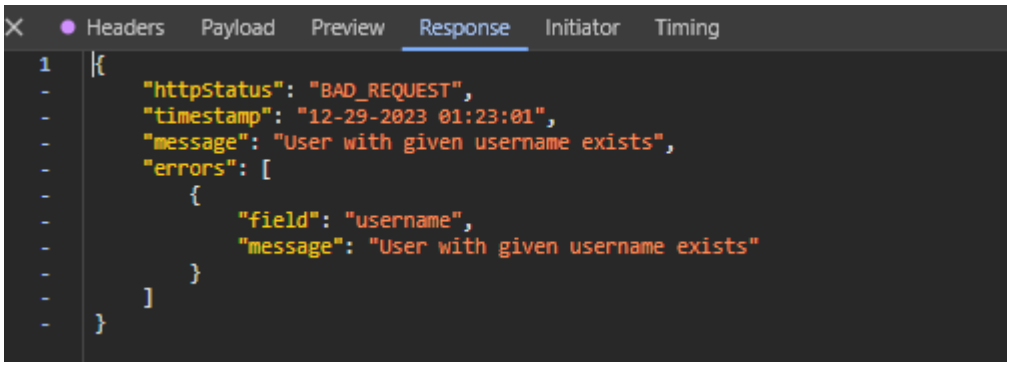
Rysunek 3.20. Właściwie zaimplementowany mechanizm zwracania informacji o błędnych danych logowania.
Źródło: opracowanie własne

Podczas rejestracji użytkownika, mechanizm wyświetlania błędu jest zaimplementowany niepoprawnie, tak jak zostało to zaprezentowane na rysunku 3.21. Atakujący może dokonywać enumeracji nazw użytkownika, sprawdzając czy podany login istnieje już w systemie.



Rysunek 3.21. Błędna implementacja mechanizmu walidacji nazwy użytkownika, ujawniająca jego istnienie w systemie.
Źródło: opracowanie własne

Dodatkowo całość ataku może być skutecznie zautomatyzowana, korzystając z odpowiedzi zwracanej w żądaniu, zaprezentowanej na rysunku 3.22.



Rysunek 3.22. Odpowiedź od serwera o istnieniu użytkownika z podaną nazwą.
Źródło: opracowanie własne

Wykorzystanie podatności

Podatność ujawnia jedynie istnienie użytkownika o podanej nazwie i sama w sobie nie stanowi zagrożenia w bezpieczeństwie aplikacji. Natomiast przy braku zabezpieczeń w mechanizmach logowania, może być skutecznie wykorzystana jako część ataku blokada usług. Atakujący najpierw odkryje istniejące nazwy użytkowników przy pomocy panelu rejestracji, a następnie może wykonać zmasowany atak siłowy.

Rekomendowane rozwiązanie

Samo ujawnienie, że nazwa istnieje nie wymaga naprawy. Jest to powszechny mechanizm dla użytkowników, stosowany np. przy zakładaniu konta Google. Problem pojawia się gdy łączy się on z brakiem mechanizmów ochrony przy wykorzystaniu uzyskanej w ten sposób nazwy użytkownika, jak np. atak siłowy.

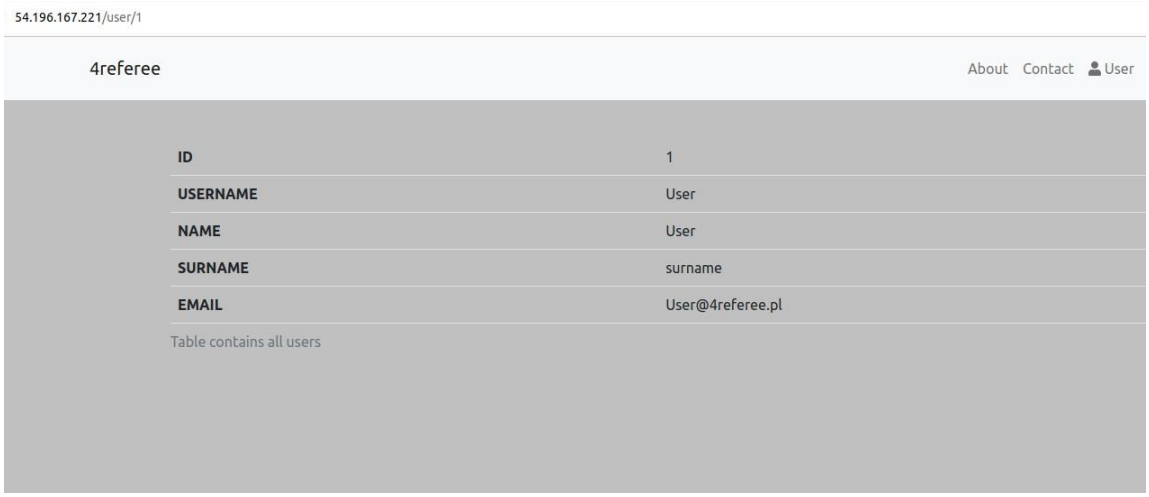
3.9. Wykryta podatność Insecure Direct Object Reference podczas pobierania danych o użytkowniku

Opis podatności

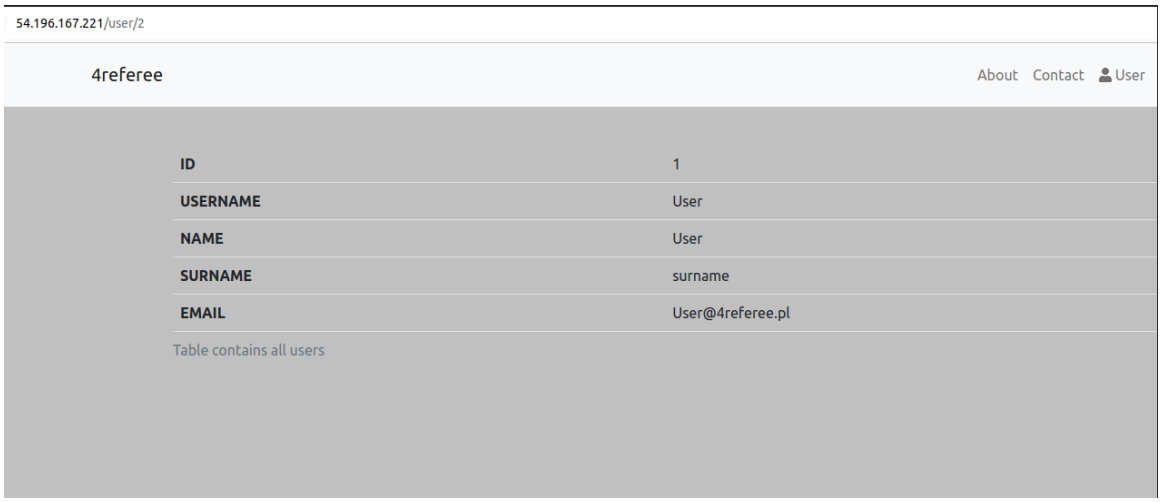
Dane o użytkowniku pobierane są z wykorzystaniem adresu url: */api/users/{id}*. Odwołanie się do tego adresu dla użytkownika o zwykłej roli powinno umożliwiać pobranie tylko danych o sobie. Tak się jednak nie dzieje. Punkt końcowy pozostaje niezabezpieczony na nieautoryzowany dostęp. Atakujący, odpowiednio manipulując przesyłanym żądaniem, poprzez podmianę parametru id może uzyskać dane o innym użytkowniku.

Wykorzystanie podatności

Domyślnie użytkownikowi wyświetlane są dane o nim samym i za pomocą adresu URL w przeglądarce, nie ma możliwości wysłania zapytania na adres serwera z innym id, gdyż ten jest automatycznie ustawiany po stronie klienckiej. Na poniższych zrzutach ekranu z aplikacji (rysunek 3.23), (rysunek 3.24) widać, że niezależnie czy zapytanie wysłane jest z parametrem id 1 lub 2, wyświetlane są dane użytkownika o id = 1.

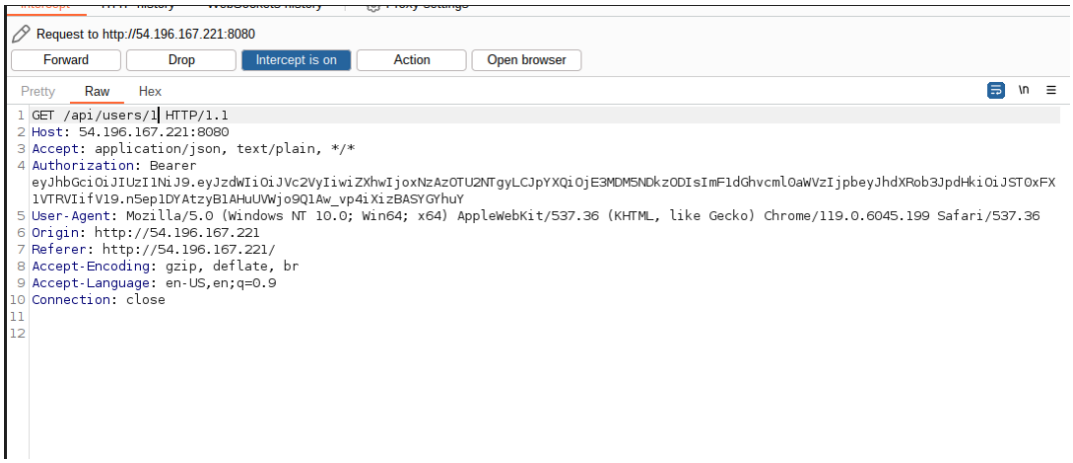


Rysunek 3.23. Zwrócone informacje o użytkowniku o id = 1.
Źródło: opracowanie własne



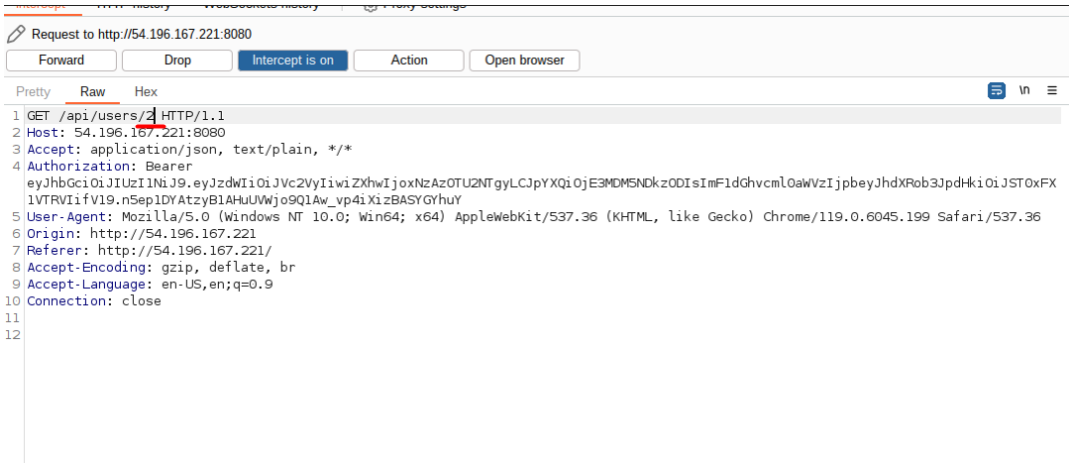
Rysunek 3.24. Zwrócone informacje o zalogowanym użytkowniku o id=1, pomimo próby manipulacji adresem URL w przeglądarce.
Źródło: opracowanie własne

W celu wykorzystania podatności, skorzystano z oprogramowania Burp, które z użyciem narzędzia proxy, przejęło wysłane żądanie do serwera. Działanie to zostało zaprezentowane na rysunku 3.25.



Rysunek 3.25. Przejęte żądanie wykonane przez widok aplikacji dla użytkownika o id=1.
Źródło: opracowanie własne

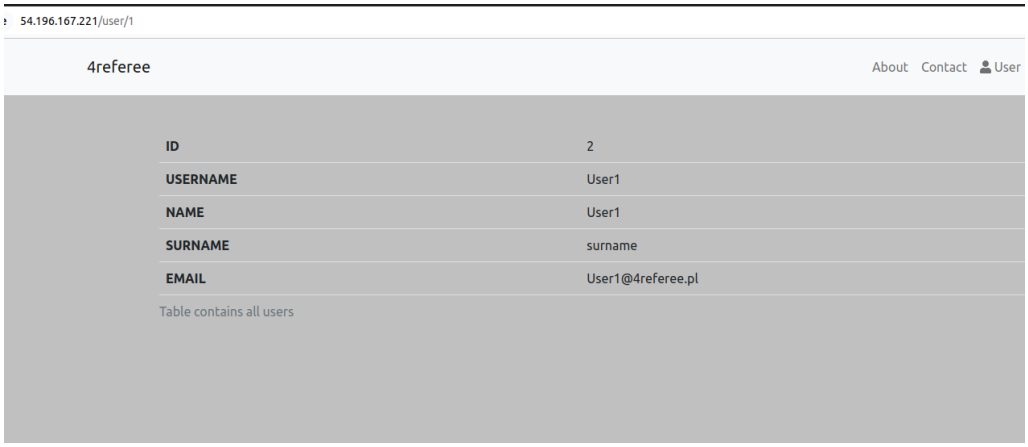
Następnie zgodnie z zrzutem ekranu, widocznym na rysunku 3.26, zmieniono parametr id w żądaniu został na identyfikator innego użytkownika w systemie.



Rysunek 3.26. Żądanie ze zmienioną wartością id na wartość 2.
Źródło: opracowanie własne

Efekt przeprowadzonego ataku

Pomimo wysłania żądania na adres *user/1*, serwer po przechwyceniu żądania przez narzędzie Burp, otrzymał prośbę o zwrócenie danych użytkownika o id = 2. Nie mając właściwie zaimplementowanej walidacji, spełnił żądanie atakującego, zwracając dane innego użytkownika. Rezultat ataku widoczny jest na rysunku 3.27.



Rysunek 3.27. Efekt skutecznie przeprowadzonego ataku - wyświetlenie danych użytkownika user1 będąc zalogowanym jako user.
Źródło: opracowanie własne

Rekomendowane rozwiązanie

Konieczne jest poprawienie mechanizmu autoryzacji. Identyfikator użytkownika powinien być pobierany z serwera lub dokładnie sprawdzony, jeśli musi być przekazywany przez osobę korzystającą z aplikacji. Zezwolenie na manipulowanie w żądaniach wartością id, która jest liczbą, jest niebezpieczne. Jeśli konieczne jest przekazywanie tego parametru, lepiej zastosować złożony, losowy identyfikator UUID.

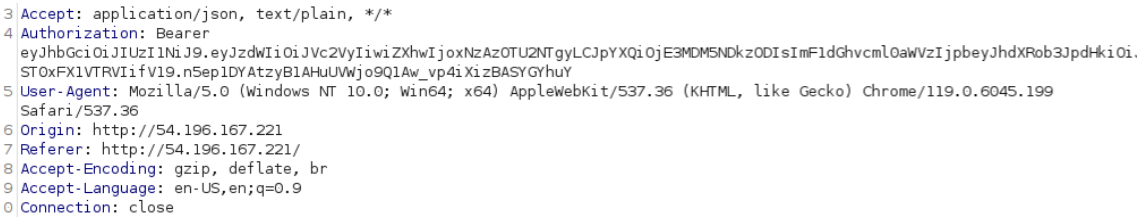
3.10. Wykryty brak dezaktywacji JWT Token po wylogowaniu użytkownika

Opis podatności

Token sesyjny po wylogowaniu się użytkownika nie jest dezaktywowany po zakończeniu sesji. Oznacza to, że system usuwa token z pamięci aplikacji, natomiast ten w dalszym ciągu jest aktywny. Atakujący przechwytując go może cały czas wykonywać nieautoryzowane akcje w imieniu nieświadomej ofiary.

Wykorzystanie podatności

Przejęcie tokenu JWT zasymulowano poprzez uzyskanie go z nagłówka wykonanego żądania. Uzyskany token widoczny jest na rysunku 3.28. Następnie klikając na przycisk „logout” wylogowano się z aplikacji.



Rysunek 3.28. Uzyskanie JWT Token z nagłówka żądania.
Źródło: opracowanie własne

Efekt przeprowadzonego ataku

Wykorzystując przechwycony token, przy użyciu oprogramowania Postman, zostało wykonane żądanie na adres api/videos/1, autoryzując je nieunieważnionym tokenem wylogowanego użytkownika. Efektem takich działań było zwrócenie danych o klipach video, pomimo wylogowania się z aplikacji, co zostało pokazane na rysunku 3.29.

Rysunek 3.29. Otrzymana odpowiedź z wykonanego żądania, autoryzowanego tokenem wylogowanego użytkownika.
Źródło: opracowanie własne

Rekomendowane rozwiązanie

Należy zaimplementować rozwiązanie, które będzie unieważniało token po wylogowaniu. Wymagany do tego będzie punkt końcowy odpowiadający za obsługę tego procesu. Innym z rozwiązań jest wprowadzenie czarnej listy (ang. blacklist), do której po wylogowaniu będzie dodawany token. Następnie przed każdym zapytaniem dokonywane będzie sprawdzenie, czy podany token istnieje na czarnej liście. Jeśli tak autoryzacja zakończy się niepowodzeniem.

4. Podsumowanie

W powyższej pracy magisterskiej został zrealizowany cel pracy – stworzono aplikację internetową do przetwarzania plików multimedialnych. Następnie przeprowadzono szeroki przegląd zabezpieczeń i występujących podatności, obejmujące analizę kodu źródłowego, testy z zakresu bezpieczeństwa sieciowego i aplikacji webowych. Zagadnienia występujące w projekcie zostały opisane w części teoretycznej, a następnie wykorzystane w części praktycznej. Luki w bezpieczeństwie oprogramowania zostały wykryte na każdym z etapów – począwszy od analizy kodu, po testy penetracyjne. Niektóre z błędów takie jak np. niebezpieczne bezpośrednie odwołanie do obiektu, czy atak siłowy powtarzały się na kilku z etapów. Inne z kolei jak np. atak człowiek w środku, czy atak DoS, były możliwe do wykrycia tylko na jednym z nich. Biorąc pod uwagę powyższe spostrzeżenia, należy wnioskować, że nie da się odpowiednio chronić aplikacji, opierając się tylko na jednym z elementów bezpiecznego procesu wytwarzania oprogramowania. Bazując tylko na weryfikacji kodu pod względem bezpieczeństwa, statycznej analizie kodu czy samych testach penetracyjnych, przeprowadzając audyt bezpieczeństwa mogą zostać niewykryte. Zapewnianie odpowiedniego poziomu zabezpieczeń, powinno być integralną częścią każdego etapu w procesie wytwarzania oprogramowania - projektu aplikacji, tworzenia kodu źródłowego, testowania, wdrożenia, utrzymania oraz monitoringu.

Twórcy oprogramowania powinni pamiętać, że zgoda na udostępnianie danych przez użytkowników, wiąże się z ich zaufaniem do aplikacji, ale również z odpowiedzialnością. Potencjalne ataki czy wycieki danych, mogą w znaczący sposób naruszyć wiarygodność oprogramowania, dlatego też o odpowiedni poziom zabezpieczeń należy dbać na każdym etapie działalności organizacji. Odkładanie działań z zakresu bezpieczeństwa oprogramowania w czasie, powoduje rosnącą ilość luk występujących w aplikacji, tym samym co raz bardziej narażając organizację na ataki. Dlatego też zaczynając nowy projekt, należy korzystać z rozwiązań bezpiecznych, nie zawierających w sobie podatności, edukować programistów przeprowadzających oceny kodu, a same testy powierzać osobom kompetentnym w zakresie bezpieczeństwa. Po wdrożeniu aplikacji na produkcję, powinna ona zawierać niezbędne elementy monitorujące ruch, a także logi pochodzące z aplikacji. Tylko spełnienie powyższych wymagań, pozwoli na unikanie podatności i ataków na aplikację.

Zrealizowanie pracy umożliwiło autorowi poszerzenie swojej wiedzy z zakresu tworzenia bezpiecznych aplikacji internetowych oraz poznanie nowych metod i narzędzi do weryfikacji istniejących zagrożeń w oprogramowaniu. Praca pozwoliła na rozwinięcie wiedzy o ochronie i testowaniu oprogramowania na każdym z etapów jego wytwarzania, dając jej twórcy kompetencje do kompleksowego tworzenia bezpiecznych aplikacji webowych.

Spis rysunków

Spis listingów

Bibliografia

W opracowaniu korzystano z następujących pozycji literaturowych:

[1] Conklin L., Robinson G.:*OWASP Code Review Guide v2*, 2017

[2] Introduction to dependency mechnism https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html, dostęp 03.12.2023

[3] Threat modeling for drivers, https://learn.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers dostęp 04.12.2023

[4] Coldwind Gynavel, Sajdak Michał, Marek Rzepecki, et al.: *Wprowadzenie do bezpieczeństwa IT*, Tom 1, Securitum 2023

[5] Badurski Dariusz Bińkowski Henryk Boryn Paweł, et al:. *Słownik terminów z zakresu bezpieczeństwa narodowego*, wydanie szóste, Warszawa 2008

[6] Artur Szeleszyński, Anna Wojaczek *The security of IT resources protected by passwords, Journal of sciecnce of the military academy of land forces,* 2015

[7] Jorge Eliecer Gómez Gómez, Sebastian Ricardo Cardenas, Fabian Sanchez Ruiz, *Patient identification system based on NFC and blockchain technology,* Universidad de Cordoba, Columbia 2023

[8] Steven Palmer:. *Web Application Vulnerabilities: Detect, Exploit, Prevent*, Syngress 2007

[9] Bentkowski Michał, Czyż Artur, Janicki Rafał, et al.: *Bezpieczeństwo Aplikacji Webowych, Securitum* 2019

[10] Fundacja Instytut Cyberbezpieczeństwa.: *Czym jest Atak DoS* https://instytutcyber.pl/artykuly/atak-dos/ dostęp 11.12.2023

[11] Daniel Missler.: https://github.com/danielmiessler/SecLists/blob/master/Passwords/xato-net-10-million-passwords-1000000.txt dostęp 14.12.2023

[12] Loscri V., Mitton Nathalie, Staddon Edward.: *Attack Categorisation for IoT Applications in Critical Infrastructures, a Survey,* Lille 08.2021

[13] Lyon Gordon *Opis programu Nmap*:. https://nmap.org/man/pl/, dostęp 17.12.2023

[14] The Ettercap Project.: https://www.ettercap-project.org/, dostęp 19.12.2023

[15] The Wireshark Foundation.: *Wireshark User's Guide* https://www.wireshark.org/docs/wsug_html/, dostęp 20.12.2023

[16] Snyk.: *Snyk Documentation,* https://docs.snyk.io/, dostęp 21.12.2023

[17] PortSwigger.: *Burp Suite Documentation,* https://portswigger.net/burp/documentation, dostęp 22.12.2023

[18] Bentkowski Michał.: *OWASP Top Ten 2021: A05: Security Misconfiguration. Przegląd przypadków* https://sekurak.pl/owasp-top-ten-2021-a05-security-misconfiguration-przeglad-przypadkow/ dostęp 28.12.2023

[19] Snyk.: *Priority Score* https://docs.snyk.io/scan-using-snyk/find-and-manage-priority-issues/priority-score, dostęp 30.12.2023

[20] The OWASP Foundation.: *Introduction* https://owasp.org/Top10, dostęp 06.01.2024

[21] Gradle.: *Dependency Management Terminology* https://docs.gradle.org/current/userguide/dependency_management_terminology.html , dostęp 08.01.2024

[22] Andrew Hoffman:. *Bezpieczeństwo nowoczesnych aplikacji internetowych, przewodnik po zabezpieczeniach*, Helion 2020

[23] Kohnfelder Loren:. *Designing Secure Software: a guide for Developers*, No Strach Press 2021

[24] H2 Database:. *H2 Database Engine* https://www.h2database.com/html/main.html, dostęp 09.01.2024

[25] Spring Framework:. *Spring Framework*, https://spring.io/projects/spring-framework/, dostęp 09.01.2024

[26] Spring Boot:. *Spring Boot,* https://spring.io/projects/spring-boot/ , dostęp 09.01.2024

[27] Spring Data:. *Spring Data* https://spring.io/projects/spring-data/, dostęp 09.01.2024

[28] Spring Security:. *Spring Security*, https://spring.io/projects/spring-security/, dostęp 10.01.2024

[29] Docker:. *Docker overview* https://docs.docker.com/get-started/overview/, dostęp 10.01.2024

[30] Amazon Ec2:. *Amazon EC2,* https://aws.amazon.com/ec2/?nc2=h_ql_prod_fs_ec2, dostęp 11.01.2024

[31] Amazon S3:. *Amazon S3,* https://aws.amazon.com/s3/?nc2=h_ql_prod_st_s3, dostęp 11.01.2024

[32] MDN Web Docs. *Cross-Origin Resource Sharing (CORS)*. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS, dostęp 13.01.2023

[33] The MITRE Organization. *CWE-352: Cross-Site Request Forgery (CSRF)*. https://cwe.mitre.org/data/definitions/352.html, dostęp 13.01.2023

[34] Farhana Sethi1:. *Automating Software Code Deployment using Continuous Integration and Continuous Delivery pipeline for business intelligence solutions, journalijsr 2020*

[35] National Vulnerability Database:. *Vulnerability*

W pracy magisterskiej zostały przeprowadzone szerokie testy bezpieczeństwa autorskiej aplikacji do przetwarzania plików multimedialnych. Praca obejmuje przegląd kodu źródłowego pod kątem bezpieczeństwa, testy bezpieczeństwa sieciowego oraz testy z zakresu bezpieczeństwa aplikacji internetowych. Niezbędne zagadnienia zostały wprowadzone w części teoretycznej, a następnie wykorzystane w części praktycznej. Część praktyczna zawiera wprowadzenie aplikacji – opis architektury i przedstawienie działania oprogramowania. Następnie w części analizy kodu pod kątem bezpieczeństwa, przedstawione są listingi wraz z ich analizą - opisem podatności jak i rekomendowanymi działaniami, mającymi poprawić zabezpieczenia w pokazanych fragmentach kodu. Kolejnym etapem części praktycznej było zaprezentowanie statycznej analizy kodu dokonanej przy wykorzystaniu narzędzia Snyk. Dalej zostały wykonane ataki takie jak: blokada usług, człowiek w środku, atak siłowy, niebezpieczne bezpośrednie odwołanie do obiektu, czy ujawnienie błędów konfiguracyjnych.