TESTOWANIE PENETRACYJNE APLIKACJI WEBOWEJ

JULIA TRZECIAKIEWICZ

Uniwersytet Kazimierza Wielkiego
Wydział Informatyki
ul. M. Kopernika 1, 85-064 Bydgoszcz

Testowanie penetracyjne aplikacji webowej, na przykładzie OWASP Juice Shop, pozwala sprawdzić, jak łatwo można znaleźć i wykorzystać różne luki bezpieczeństwa w popularnych systemach internetowych. W tym badaniu skupiono się na najczęstszych typach zagrożeń, takich jak błędy w kontroli dostępu, niewłaściwe szyfrowanie, podatności na wstrzykiwanie kodu czy słaba konfiguracja serwera. Do analizy wykorzystano zarówno narzędzia automatyczne, jak i ręczne metody testowania, co pozwoliło na odkrycie wielu typowych problemów, od możliwości przejęcia konta, przez wyciek poufnych danych, aż po błędy w logowaniu i monitorowaniu. Wyniki pokazują jak wiele różnych podatności może zawierać aplikacja, która na pierwszy rzut oka wygląda zwyczajnie. Warto jednak podkreślić, że w przypadku Juice Shop wszystkie te podatności zostały celowo zaimplementowane, aby umożliwić naukę i praktyczne ćwiczenie wykrywania zagrożeń, co czyni tę aplikację świetnym narzędziem do zdobywania doświadczenia w testowaniu bezpieczeństwa.

1. WSTĘP

Bezpieczeństwo aplikacji webowych to jedno z kluczowych wyzwań współczesnej informatyki, szczególnie w kontekście rosnącej liczby usług internetowych i przetwarzania wrażliwych danych online. Coraz częstsze ataki na aplikacje webowe pokazują, że nawet popularne i szeroko stosowane rozwiązania mogą być podatne na różnorodne zagrożenia. W odpowiedzi na te wyzwania powstały narzędzia i projekty edukacyjne, które umożliwiają praktyczne ćwiczenie wykrywania i wykorzystywania typowych błędów bezpieczeństwa, jednym z najbardziej rozpoznawalnych jest OWASP Juice Shop, celowo wyposażony w szeroki wachlarz podatności zgodnych z OWASP Top 10. W literaturze podkreśla się, że testy penetracyjne stanowią niezbędny element procesu rozwoju oprogramowania, a środowiska takie jak Juice Shop pozwalają zarówno początkującym, jak i zaawansowanym testerom na doskonalenie umiejętności w bezpiecznych warunkach. Głównym celem niniejszej pracy jest analiza 10 najpopularniejszych podatności na przykładzie OWASP Juice Shop oraz przegląd skutecznych metod ich wykrywania i zabezpieczania aplikacji webowych. Przeprowadzone testy potwierdzają, że nawet celowo uproszczone środowisko może skutecznie uczyć rozpoznawania i eliminowania najważniejszych zagrożeń dla bezpieczeństwa aplikacji webowych [1][2].

2. MATERIAŁY I METODY

Jak wyżej wspomniano do przeprowadzenia testów penetracyjnych wykorzystano aplikację OWASP Juice Shop, która jest publicznie dostępnym, celowo podatnym projektem edukacyjnym. Instalację środowiska testowego rozpoczęto od pobrania i zainstalowania Node.js w wersji zgodnej z zaleceniami projektu. Następnie, korzystając z polecenia git clone https://github.com/juice-shop/juice-shop.git --depth 1, pobrano repozytorium aplikacji na lokalny komputer, po czym przeprowadzono instalację zależności za pomocą npm install oraz uruchomiono aplikację poleceniem npm start [1][6].

Do przechwytywania i modyfikowania ruchu sieciowego wykorzystywano narzędzie Burp Suite Community Edition, które zostało pobrane i zainstalowane zgodnie z oficjalną instrukcją producenta. Aby umożliwić analizę żądań HTTP/HTTPS przesyłanych pomiędzy przeglądarką a aplikacją, skonfigurowano przeglądarkę Google Chrome do pracy przez proxy Burp Suite, wykorzystując rozszerzenie FoxyProxy4. Dodatkowo, zaimportowano certyfikat CA Burp Suite do systemu Windows i przeglądarki, co pozwoliło na pełną inspekcję ruchu szyfrowanego [5].

Pobrano również Nmap i SQLmap. Nmap to zaawansowane narzędzie służące do skanowania sieci i wykrywania otwartych portów na wskazanych hostach. Umożliwia szybkie rozpoznanie usług dostępnych na serwerze, co jest niezbędnym etapem wstępnej analizy bezpieczeństwa. SQLmap to otwartoźródłowe narzędzie automatyzujące wykrywanie i eksploatację podatności typu SQL Injection w aplikacjach webowych. Umożliwia testowanie parametrów przesyłanych w żądaniach HTTP pod kątem podatności na iniekcję SQL, identyfikację typu i wersji bazy danych, pobieranie danych, a nawet przejmowanie kontroli nad serwerem bazodanowym [3][4].

Scenariusze testowe obejmowały różne możliwe zachowania i próby wykorzystania podatności takich jak Broken Access Control, Cryptographic Failures czy Injection, a także analizę odpowiedzi serwera na zmodyfikowane żądania. Wszystkie działania były dokumentowane, a wyniki porównywano z wytycznymi OWASP Top 10. Metodyka opierała się na ogólnie przyjętych standardach testowania bezpieczeństwa aplikacji webowych oraz oficjalnej dokumentacji OWASP Juice Shop.

3. SZCZEGÓŁY TESTOWANIA

Przegląd funkcjonalności aplikacji i jej architektury

Zaprojektowana jest jako pełnoprawny sklep internetowy, oferując funkcjonalności typowe dla e-commerce, takie jak: przeglądanie produktów, wyszukiwanie, składanie zamówień, rejestracja i logowanie użytkowników, zarządzanie kontem, wystawianie opinii, kontakt z obsługą oraz system koszyka zakupowego. Dodatkowo aplikacja zawiera system wyzwań, które polegają na odnajdywaniu i wykorzystywaniu celowo wprowadzonych podatności.

Architektura aplikacji opiera się w całości na języku JavaScript i TypeScript. Frontend zbudowany jest w technologii Angular (Single Page Application) z wykorzystaniem Angular Material i Flex-Layout, zapewniając nowoczesny, responsywny interfejs użytkownika. Backend oparty jest na Node.js i frameworku Express, a komunikacja między frontendem i backendem odbywa się poprzez RESTful API. Dane przechowywane są w lekkiej bazie SQLite, a do obsługi operacji na bazie wykorzystywane są Sequelize oraz finale-rest. Dodatkowo aplikacja korzysta z MarsDB, będącej odmianą bazy NoSQL kompatybilnej z MongoDB.

Kluczowe punkty wejścia w aplikacji OWASP Juice Shop obejmują zarówno klasyczne formularze webowe, jak i publiczne endpointy REST API, przez które użytkownicy oraz potencjalni atakujący mogą komunikować się z systemem:

Formularze webowe:

1. Formularz logowania i rejestracji użytkownika
2. Formularz zmiany hasła i zarządzania kontem
3. Formularz wyszukiwania produktów
4. Formularz składania zamówienia i płatności
5. Formularz wystawiania opinii o produktach
6. Formularz kontaktowy do obsługi klienta

Publiczne endpointy API:

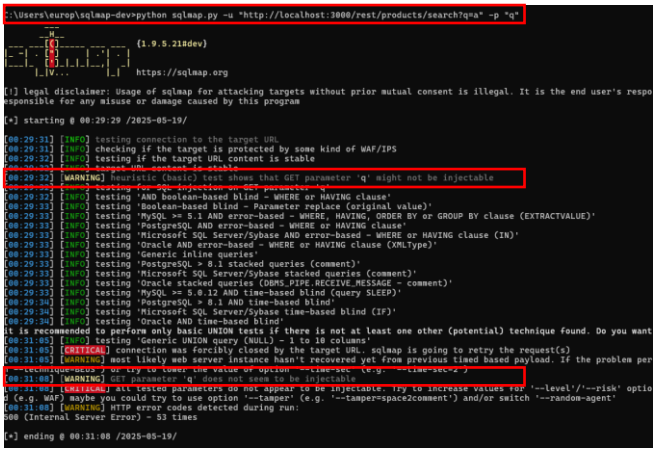
1. /rest/user/login – logowanie użytkownika
2. /rest/user/register – rejestracja nowego użytkownika
3. /rest/products/search – wyszukiwanie produktów
4. /rest/basket – zarządzanie koszykiem zakupowym
5. /rest/feedback – przesyłanie opinii
6. /api/Challenges – pobieranie listy wyzwań i ich statusów3
7. /snippets oraz /snippets/<challengeKey> – pobieranie fragmentów podatnego kodu

Te punkty wejścia są kluczowe z punktu widzenia bezpieczeństwa, gdyż właśnie przez nie aplikacja przyjmuje dane od użytkowników i komunikuje się z bazą danych oraz innymi komponentami backendu. W praktyce większość wyzwań i podatności w Juice Shop związana jest z nieprawidłową walidacją danych wejściowych w tych miejscach, co czyni je priorytetowymi celami podczas testów penetracyjnych.

SQLmap

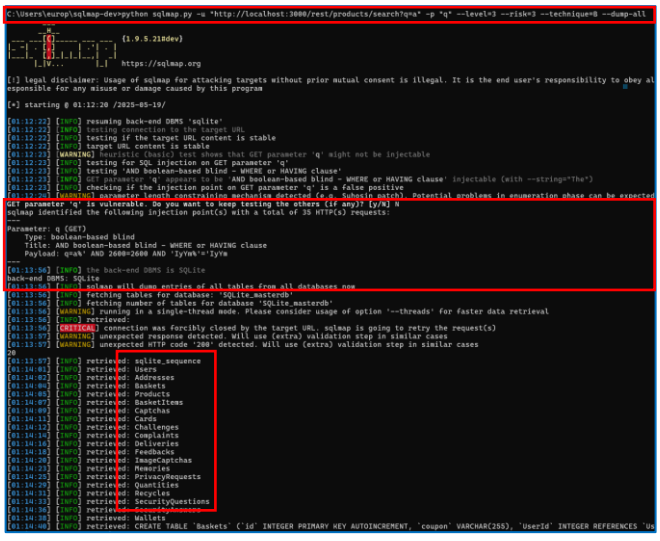
W trakcie realizacji testów bezpieczeństwa aplikacji OWASP Juice Shop przeprowadzono proces iteracyjnego doskonalenia komendy wykorzystywanej w narzędziu SQLmap do automatycznego wykrywania oraz eksploatacji podatności typu SQL Injection. Początkowe próby z podstawowymi parametrami nie przyniosły oczekiwanych rezultatów, ponieważ SQLmap nie wykrywał podatnych

parametrów lub nie był w stanie potwierdzić obecności podatności.



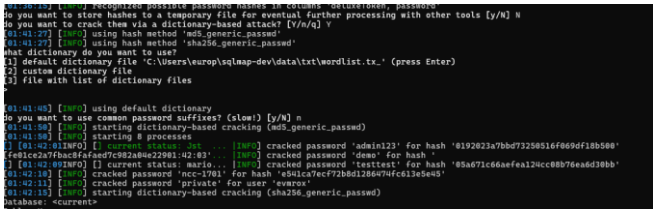
Rysunek. 1 Pierwsza nieudana próba.

Analiza komunikatów zwracanych przez narzędzie pozwoliła na stopniowe rozszerzanie zakresu testów poprzez dodawanie opcji takich jak --level=3, --risk=3, a także wskazanie konkretnego parametru (-p "q") oraz endpointu (/rest/products/search?q=a). W miarę postępu testów SQLmap rozpoznał, że backendową bazą danych jest SQLite, a parametr q jest podatny na atak typu UNION-based SQL Injection.



Rysunek. 2 Próba zakończona sukcesem.

Dzięki odpowiedniej konfiguracji narzędzia udało się nie tylko potwierdzić podatność, ale również przeprowadzić pełny atak, polegający na pobraniu struktury oraz zawartości bazy danych aplikacji. Wśród pozyskanych informacji znalazły się m.in. dane użytkowników, hashe haseł, odpowiedzi bezpieczeństwa oraz inne wrażliwe rekordy. Co istotne, SQLmap umożliwił automatyczne łamanie wykrytych hashy, a w tym przypadku wykorzystano zestaw popularnych hashy. Pozwoliło to na uzyskanie jawnych haseł do części kont użytkowników.



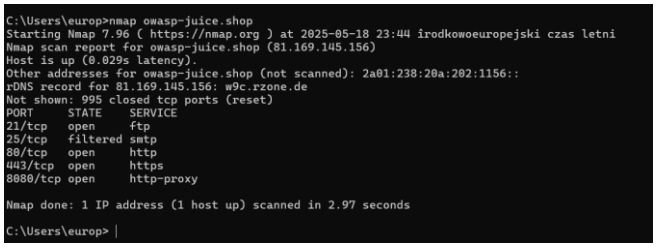
Rysunek.3 Złamane hasła przez SQLmap.

Cały proces pokazał, jak istotne jest świadome i elastyczne wykorzystanie narzędzi automatycznych, analiza zwracanych komunikatów oraz dostosowywanie parametrów testu do specyfiki testowanej aplikacji. Ostatecznie udało się w pełni zautomatyzować wykrycie i eksploatację podatności SQL Injection, co w środowisku rzeczywistym mogłoby prowadzić do poważnego incydentu bezpieczeństwa.

Rysunek.4 Przykładowa tabela z bazy danych o nazwie Memories.

Nmap

Wstępna analiza bezpieczeństwa każdej aplikacji webowej powinna rozpoczynać się od rozpoznania środowiska sieciowego, w którym działa testowany system. Jednym z najważniejszych narzędzi wspierających ten etap jest Nmap, zaawansowany skaner sieciowy, który umożliwia szybkie i skuteczne wykrywanie otwartych portów oraz usług dostępnych na serwerze. Wyniki uzyskane dzięki Nmap pozwalają na ocenę powierzchni ataku oraz identyfikację potencjalnych podatności, które mogą być wykorzystane w kolejnych fazach testów penetracyjnych.



Rysunek.5 Wyniki skanowania portów Nmap.

W wyniku skanowania Nmap zidentyfikowano kilka otwartych portów, m.in. 21 (FTP), 80 (HTTP), 443 (HTTPS), 8080 (HTTP-proxy) oraz porty filtrowane, takie jak 25 (SMTP). Każdy z tych portów reprezentuje usługę, która, jeśli jest nieaktualna, źle skonfigurowana lub niepotrzebna, może stanowić potencjalne zagrożenie. Szczególnie istotne są porty HTTP i HTTPS, ponieważ wskazują na obecność serwera WWW, będącego głównym celem dalszych testów aplikacji webowej. Dodatkowe usługi, takie jak FTP czy proxy, mogą zwiększać powierzchnię ataku i wymagają szczegółowej analizy pod kątem podatności oraz zasadności ich utrzymywania.
Wynik skanowania Nmap stanowi fundament do dalszych działań, pozwala na ukierunkowanie testów na konkretne usługi, wykrywanie przestarzałych komponentów oraz analizę konfiguracji bezpieczeństwa środowiska. Prawidłowa interpretacja tych danych umożliwia skuteczne wykrywanie i minimalizowanie ryzyka związanego z podatnościami zarówno na poziomie infrastruktury, jak i samej aplikacji webowej.

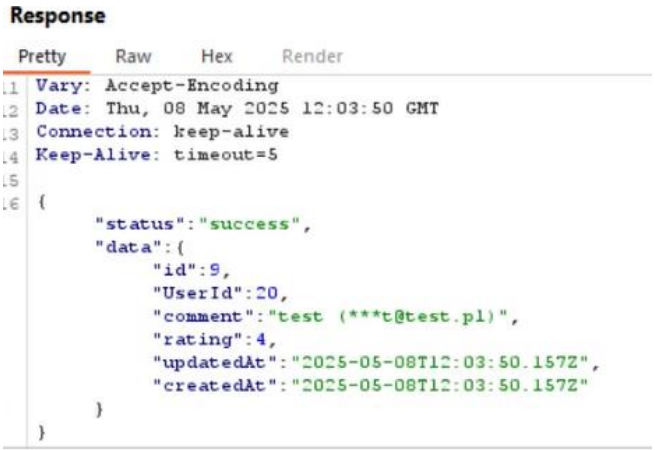
OWASP TOP 10

1. A01 Broken Access Control (Naruszenie kontroli dostępu)

To jedna z najpoważniejszych i najczęściej spotykanych podatności w aplikacjach webowych, polegająca na umożliwieniu użytkownikom wykonywania operacji wykraczających poza ich uprawnienia. Skutkiem mogą być nieautoryzowane działania, takie jak dostęp do cudzych danych, ich modyfikacja czy podszywanie się pod innych użytkowników.

W ramach testów OWASP Juice Shop zademonstrowano typowy scenariusz tego typu podatności. Po utworzeniu konta i zalogowaniu się, użytkownik przeszedł do sekcji opinii. Przy użyciu narzędzia Burp Suite przechwycono żądanie HTTP związane z publikacją opinii i zmodyfikowano pole userId na identyfikator innego użytkownika. Po wysłaniu zmienionego żądania aplikacja zaakceptowała opinię jako napisaną przez wskazaną osobę, co potwierdziło brak właściwej weryfikacji uprawnień po stronie serwera.

Ten przykład pokazuje, jak łatwo można obejść mechanizmy autoryzacji, jeśli nie są one prawidłowo zaimplementowane. W praktyce prowadzi to do nadużyć i utraty zaufania do systemu. Zaleca się, aby wszelkie operacje były domyślnie zablokowane, a dostęp przyznawany wyłącznie na podstawie jasno określonych ról i uprawnień. Mechanizmy kontroli dostępu powinny być scentralizowane, konsekwentnie stosowane oraz regularnie testowane. Dodatkowo warto wdrożyć logowanie prób nieautoryzowanego dostępu i ograniczenia liczby żądań, aby utrudnić ataki zautomatyzowane.



Rysunek.6 UserId wynosi 23, a opinie przesłano z id=20.

2. A02 Cryptographic Failures (Błędy kryptograficzne, dawniej Sensitive Data Exposure)

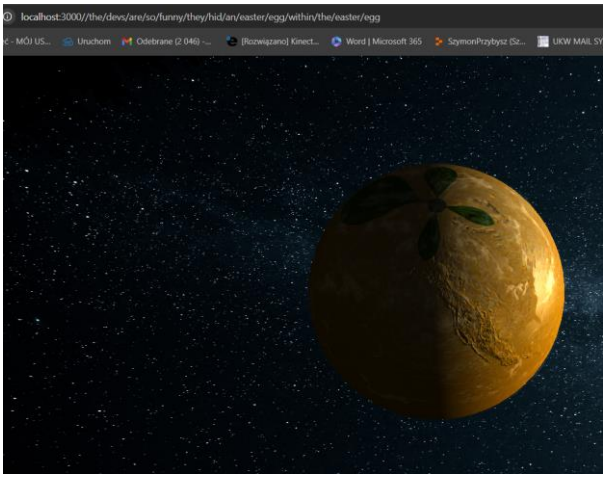
To podatność obejmująca wszelkie przypadki niewłaściwego stosowania mechanizmów kryptograficznych w aplikacjach. Zamiast skupiać się tylko na skutkach, jak wyciek danych. Podkreśla się tu źródło problemu: brak szyfrowania, użycie przestarzałych algorytmów, nieprawidłowe zarządzanie kluczami czy traktowanie kodowania (np. Base64) jako zabezpieczenia.

W OWASP Juice Shop podatność ta została zademonstrowana w wyzwaniu „Weird Crypto”, gdzie użytkownik ma wskazać błędnie użyty algorytm. Wystarczy

w formularzu opinii wpisać „base64”, by zaliczyć zadanie, ponieważ Base64 nie zapewnia żadnego bezpieczeństwa jest jedynie kodowaniem, a nie szyfrowaniem. Takie błędy prowadzą do sytuacji, w których poufne dane mogą być łatwo przechwycone lub odczytane przez osoby niepowołane.

Następnie wyzwanie „Nested Easter Egg”. Na początku użytkownik wpisuje w adresie przeglądarki %2500.md, co jest zakodowaną wersją znaku null (%00), pozwalającą na dostęp do ukrytego pliku tekstowego. Plik ten zawiera ciąg znaków zakodowany w Base64, który po zdekodowaniu ujawnia kolejną, nadal nieczytelną wiadomość. Następnie stosuje się szyfr ROT13, który odsłania właściwą treść i ścieżkę do „easter egg”.

Aby uniknąć tego typu podatności, kluczowe jest stosowanie aktualnych, silnych algorytmów szyfrujących, takich jak AES czy RSA w odpowiednich konfiguracjach. Niezbędne jest również szyfrowanie danych zarówno podczas transmisji (np. z użyciem protokołu TLS), jak i w stanie spoczynku (np. w bazach danych lub na dyskach). Równie istotne jest właściwe zarządzanie kluczami szyfrującymi, w tym ich bezpieczne przechowywanie, cykliczna rotacja oraz kontrola dostępu.



Rysunek.7 Widok po wklejeniu odszyfrowanej ścieżki.

3. A03 Injection (Iniekcje, np. SQL Injection)

Injection polega na wstrzykiwaniu złośliwych danych do aplikacji, które są następnie wykonywane przez bazę danych, interpreter lub inny komponent systemu. Do najczęstszych ataków tego typu należą SQL Injection czy XSS. Ich skutkiem może być nieautoryzowany dostęp do danych, modyfikacja informacji lub przejęcie kontroli nad aplikacją.

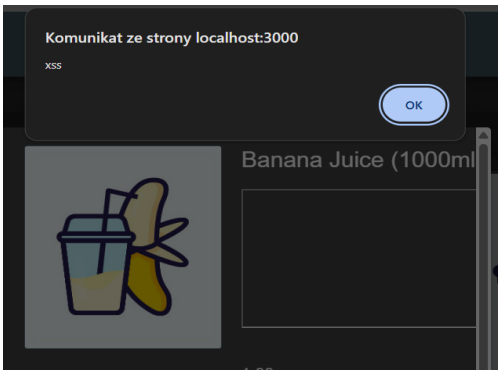
Typowym przykładem jest budowanie zapytań SQL poprzez bezpośrednie łączenie danych użytkownika z kodem zapytania. W takiej sytuacji atakujący może wstrzyknąć własny kod, np. w polu logowania, używając payloadu typu ' OR 1=1--, co pozwala zalogować się na dowolne konto, w tym przypadku administratora. W OWASP Juice Shop zadanie polega właśnie na wykorzystaniu tej luki w formularzu logowania. Brak walidacji i parametryzacji zapytań SQL umożliwia obejście mechanizmów uwierzytelniania i uzyskanie dostępu do konta administracyjnego bez znajomości hasła.

Podatność XSS pojawia się, gdy aplikacja nieprawidłowo przetwarza i wyświetla dane pochodzące od użytkownika, umożliwiając wstrzyknięcie i wykonanie złośliwego kodu JavaScript w przeglądarce innego użytkownika. W tym przypadku atakujący przesyła w polu description fragment HTML z tagiem <iframe src="javascript:alert('xss')">, który po zapisaniu i wyświetleniu przez aplikację może spowodować wykonanie kodu JavaScript.

Atak polega na przesłaniu żądania HTTP typu PUT do endpointu /api/Products/6 z odpowiednio spreparowanym polem description. W żądaniu wykorzystuje się również token administratora i nagłówek Content-Type: application/json, informujący serwer, że przesyłane dane w ciele żądania są w formacie JSON. Jeśli aplikacja nie filtruje lub nie koduje znaków specjalnych wprowadzanych przez użytkownika, kod JavaScript zostanie osadzony w odpowiedzi serwera i wykonany po stronie klienta. Może to prowadzić do kradzieży ciasteczek, przechwytywania sesji, wyświetlania fałszywych treści lub dalszych ataków na użytkowników aplikacji. Tym razem atakujący modyfikuje opis produktu, wstrzykując złośliwy kod. Jeśli aplikacja nie stosuje odpowiednich zabezpieczeń, kod zostanie wykonany w przeglądarce każdego użytkownika, który odwiedzi stronę produktu.

Aby zapobiegać tego typu podatnościom, należy stosować zapytania parametryzowane, konsekwentnie walidować i filtrować dane wejściowe oraz ograniczać uprawnienia kont

bazodanowych. Regularne testy bezpieczeństwa i użycie narzędzi automatycznych pozwalają wykryć i wyeliminować luki typu injection, zanim zostaną one wykorzystane przez atakujących.



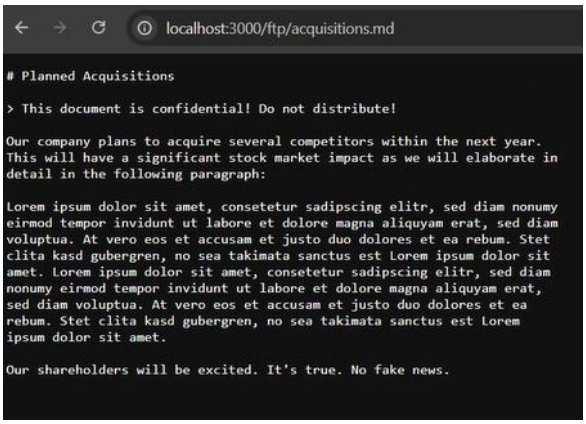
Rysunek.8 Komunikat, który potwierdza, że kod JavaScript został wstrzyknięty i wykonany.

4. A04 Insecure Design (Niebezpieczny projekt)

Insecure Design to kategoria, odnosząca się do sytuacji, gdy aplikacja już na etapie projektowania nie uwzględnia podstawowych zasad bezpieczeństwa. W efekcie powstają systemy, które są podatne na ataki niezależnie od jakości implementacji, problem leży w samej architekturze i logice działania. Podatność ta wynika z braku odpowiednich zabezpieczeń w projekcie, takich jak kontrola dostępu, właściwe zarządzanie poufnymi danymi czy ograniczenia dla użytkowników. Przykładem może być umożliwienie dostępu do plików systemowych przez manipulację URL-em lub przechowywanie wrażliwych informacji w niechronionych lokalizacjach. Takie błędy są trudne do naprawy po wdrożeniu i często wymagają zmian w całej architekturze aplikacji.

W OWASP Juice Shop podatność typu Insecure Design ilustruje wyzwanie „confidential document leak". Użytkownik, manipulując adresem URL w sekcji „About Us", może uzyskać dostęp do katalogu z poufnymi dokumentami, takimi jak acquisitions.md, które nie powinny być publicznie dostępne.

Aby uniknąć takich problemów, bezpieczeństwo powinno być uwzględniane już na etapie projektowania systemu, poprzez modelowanie zagrożeń, stosowanie zasad najmniejszych uprawnień i regularne przeglądy architektury. Tylko świadome podejście do projektowania aplikacji pozwala skutecznie zapobiegać tego typu podatnościom i chronić dane organizacji.



Rysunek.9 Plik acquisitions.md

5. A05 Security Misconfiguration (Błędna konfiguracja zabezpieczeń)

Security misconfiguration polega na niewłaściwym ustawieniu lub braku kluczowych mechanizmów ochronnych, co może prowadzić do poważnych naruszeń bezpieczeństwa nawet w dobrze zaprojektowanych systemach. Typowe przykłady to pozostawienie domyślnych kont i haseł, nieusunięcie zbędnych funkcji, brak aktualizacji, niewłaściwe uprawnienia czy umożliwienie dostępu do plików, które powinny być chronione. W aplikacjach webowych często spotyka się brak nagłówków bezpieczeństwa lub wyświetlanie szczegółowych komunikatów o błędach, które mogą ujawniać informacje o środowisku i strukturze aplikacji.

W OWASP Juice Shop podatność ta widoczna jest np. w wyzwaniu „Error Handling", gdzie użytkownik otrzymuje szczegółowy stack trace po próbie dostępu do niedozwolonego pliku. Takie informacje mogą ułatwić atakującemu dalszą eksplorację i wykorzystanie innych luk.

Kolejnym przykładem security misconfiguration jest wyzwanie „Cross-Site Imaging" na stronie Deluxe Membership. W tym przypadku podatność polega na niewłaściwym skonfigurowaniu mechanizmu pobierania zewnętrznych obrazków, co umożliwia podmianę grafik na pudełkach na obrazki z zewnętrznej domeny. Wykorzystując

podatność directory traversal i mechanizm redirect, możliwe było wstawienie własnego obrazka kota z zewnętrznego serwisu, korzystając z adresu:

```
http://localhost:3000/#/deluxe-membership?testDecal=../../../../redirect?to=https://placecats.com/neo/300/200?x=https://github.com/juice-shop/juice-shop
```

Parametr testDecal jest używany przez aplikację do określenia, jakiego obrazka użyć jako naklejki na pudełkach wyświetlanych na stronie Deluxe Membership.

../../../../redirect?to=...
Cztery ../ pozwalają „wyjść" z domyślnego katalogu obrazków aż do głównego katalogu aplikacji, gdzie dostępny jest endpoint /redirect.
/redirect?to=... to wewnętrzny mechanizm Juice Shop, który umożliwia pobranie obrazka z zewnętrznego źródła wskazanego w parametrze to.

https://placecats.com/neo/300/200
To adres obrazka kota z zewnętrznej domeny. W tym przypadku jest to przykładowy serwis z obrazkami kotów, który zostanie wyświetlony na pudełkach.

x=https://github.com/juice-shop/juice-shop
Ten dodatkowy parametr (x=...) jest wykorzystywany do obejścia mechanizmu allowlisty, która normalnie ogranicza pobieranie obrazków tylko z określonych, zaufanych źródeł. Dodanie tej wartości powoduje, że aplikacja akceptuje również obrazki z innych domen.

Rysunek.10 Zamiast logo Juice Shop widać zdjęcie kota. Wyzwanie zakończone sukcesem.

Aby ograniczyć ryzyko, należy regularnie usuwać niepotrzebne funkcje, zmieniać domyślne dane dostępowe, stosować aktualizacje, właściwie konfigurować uprawnienia oraz dbać o to, by komunikaty o błędach nie ujawniały szczegółów technicznych. Kluczowe jest także wdrażanie automatycznych audytów konfiguracji i ciągłe monitorowanie środowiska.

6. A06 Vulnerable and Outdated Components (Podatne i nieaktualne komponenty)

Vulnerable and outdated components to szósta kategoria OWASP Top 10, odnosząca się do ryzyka związanego z używaniem w aplikacjach przestarzałych lub podatnych bibliotek, frameworków i innych zależności. Współczesne aplikacje rzadko powstają wyłącznie w oparciu o własny kod, ponieważ korzystają z gotowych komponentów, które, jeśli nie są regularnie aktualizowane, mogą zawierać znane luki bezpieczeństwa.

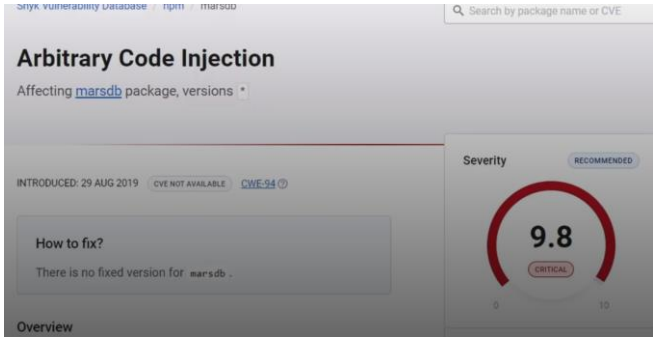
Podatność ta polega na tym, że aplikacja wykorzystuje zależności, dla których opublikowano już poprawki bezpieczeństwa lub które nie są już wspierane. Często deweloperzy nie mają pełnej świadomości, jakie wersje bibliotek są używane po stronie klienta i serwera, a brak regularnej analizy składników sprawia, że podatne komponenty mogą pozostawać niezauważone przez długi czas. Problem pogłębia się, gdy zależności pobierane są z nieoficjalnych źródeł lub nie są testowane pod kątem bezpieczeństwa.

Podczas próby pobrania pliku package.json.bak, zawierającego listę zależności aplikacji Node.js, zauważono, że aplikacja umożliwia pobieranie jedynie plików z rozszerzeniami .md oraz .pdf. Oznacza to, że bezpośredni dostęp do plików o innych rozszerzeniach, takich jak .bak, został zablokowany przez filtr rozszerzeń.

Aby obejść to ograniczenie, zastosowano technikę znaną jako null byte injection. Jest to metoda ataku polegająca na wprowadzeniu bajtu zerowego (null byte), oznaczanego jako %00 lub zakodowanego w adresie URL jako %2500, do żądania HTTP. W praktyce technika ta polegała na dodaniu do nazwy pliku sekwencji znaków, takich jak package.json.bak%2500.md. Aplikacja, niepoprawnie interpretując nazwę pliku, rozpoznała go jako posiadający dozwolone rozszerzenie .md, mimo że w rzeczywistości

chodziło o plik package.json.bak. Filtr bezpieczeństwa analizował ciąg znaków bez uwzględnienia skutków obecności bajtu zerowego, co umożliwiło pobranie zabronionego pliku.

Po uzyskaniu dostępu do pliku package.json.bak możliwe było przeanalizowanie listy bibliotek oraz komponentów wykorzystywanych przez aplikację, znajdujących się w sekcji dependencies. Pozwoliło to na ocenę potencjalnych zagrożeń wynikających z wykorzystania podatnych zależności.
W celu identyfikacji znanych podatności wykorzystano platformę Snyk, która umożliwia sprawdzenie bezpieczeństwa używanych bibliotek. Proces ten polegał na analizie każdej zależności pod kątem jej wersji oraz występujących w niej luk. Wśród wykrytych podatnych komponentów znalazły się między innymi biblioteki marsdb oraz express-jwt. Znalezione podatności mogą posłużyć jako punkt wyjścia do dalszych działań ofensywnych wobec aplikacji.



Rysunek.10 Na załączonym zrzucie ekranu przedstawiono wynik analizy podatności pakietu marsdb w bazie Snyk Vulnerability Database

Aby zapobiec tego typu zagrożeniom, niezbędne jest regularne przeprowadzanie analizy bezpieczeństwa zależności przy użyciu narzędzi takich jak Snyk, OWASP Dependency-Check czy inne specjalistyczne rozwiązania. W przypadku wykrycia podatności należy niezwłocznie zaktualizować bibliotekę do bezpiecznej wersji. Dodatkowo, zależności, które nie są wykorzystywane w projekcie, powinny zostać usunięte. Szczególną uwagę należy zwracać na źródła, z których pobierane są biblioteki, należy korzystać wyłącznie z oficjalnych repozytoriów, ponieważ korzystanie z niezweryfikowanych źródeł może prowadzić do instalacji komponentów celowo zawierających podatności, które mogą zostać wykorzystane przeciwko aplikacji.

7. A07 Identification and Authentication Failures (Błędy identyfikacji i uwierzytelniania)

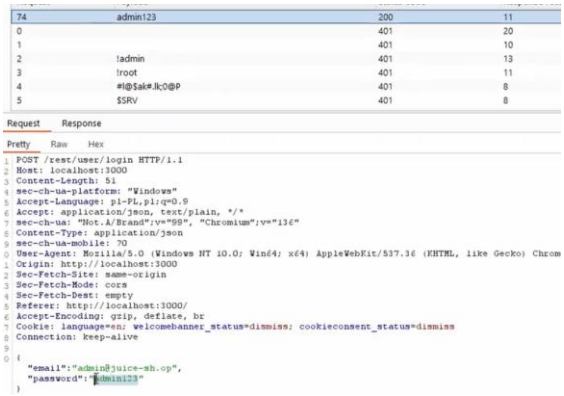
Identyfikacja i uwierzytelnianie użytkowników stanowią fundamentalne elementy systemów zabezpieczeń aplikacji internetowych. Błędy w tych obszarach mogą prowadzić do poważnych incydentów bezpieczeństwa, takich jak przejęcie kont użytkowników, nieautoryzowany dostęp do danych, czy obejście mechanizmów kontroli dostępu. Brak odpowiednich zabezpieczeń może umożliwić atakującym przeprowadzenie ataków typu brute force, polegających na systematycznym testowaniu wielu kombinacji nazw użytkowników i haseł, często pochodzących z ogólnodostępnych, skompromitowanych baz danych.

W ramach analizy bezpieczeństwa aplikacji OWASP Juice Shop przeprowadzono test odporności na ataki typu brute force. Celem testu było uzyskanie dostępu do konta administratora z wykorzystaniem znanej listy domyślnych haseł, dostępnej publicznie pod adresem:

```
https://github.com/danielmiessler/SecLists/blob/master/Passwords/cirt-default-passwords.txt
```

Do przeprowadzenia ataku wykorzystano narzędzie Burp Suite, które umożliwia przechwytywanie, analizowanie oraz modyfikowanie ruchu HTTP. Po uruchomieniu narzędzia i aktywowaniu funkcji przechwytywania żądań (Intercept On), otwarto formularz logowania w aplikacji i wprowadzono adres e-mail administratora (admin@juice-sh.op) wraz z losowym hasłem. Przechwycone żądanie typu POST, skierowane do endpointa /rest/user/login, zostało następnie przekazane do modułu Intruder w celu automatyzacji ataku.

Moduł Intruder umożliwia podmianę wartości w wybranych fragmentach żądania HTTP, co pozwala na systematyczne testowanie wielu haseł. Po wskazaniu parametru hasła jako punktu podatnego na atak i załadowaniu listy domyślnych poświadczeń, rozpoczęto symulację ataku brute force. Analizując odpowiedzi serwera, zwrócono uwagę na jedno żądanie, które zwróciło odpowiedź o innej długości niż pozostałe. Szczegółowa analiza wykazała, że odpowiedź zawierała token uwierzytelniający, co jednoznacznie wskazywało na skuteczne zalogowanie.



Rysunek.10 Na załączonym zrzucie ekranu przedstawiono wynik ataku.

Tego typu podatność należy uznać za krytyczną, gdyż umożliwia nieautoryzowane uzyskanie dostępu do konta o podwyższonych uprawnieniach, co potencjalnie może prowadzić do pełnej kompromitacji systemu. Przyczyną podatności jest zazwyczaj brak mechanizmów ograniczających liczbę nieudanych prób logowania, brak polityk wymuszających stosowanie silnych i unikalnych haseł, użycie domyślnych danych logowania oraz niewdrożenie dodatkowych warstw zabezpieczeń, takich jak uwierzytelnianie wieloskładnikowe (MFA).

W celu przeciwdziałania tego typu atakom należy zastosować szereg środków bezpieczeństwa. Kluczowe jest wdrożenie mechanizmów blokujących konto lub nakładających opóźnienia po określonej liczbie nieudanych prób logowania, uniemożliwienie korzystania z domyślnych poświadczeń, egzekwowanie polityk dotyczących tworzenia silnych haseł oraz wdrożenie wieloetapowego uwierzytelniania.

8. A08 Software and Data Integrity Failures (Naruszenia integralności oprogramowania i danych)

Integralność danych stanowi kluczowy element zapewnienia bezpieczeństwa systemów informatycznych. Odnosi się ona do gwarancji, że dane nie zostały zmodyfikowane, usunięte ani uszkodzone w sposób nieautoryzowany. Naruszenie integralności może prowadzić do szeregu negatywnych skutków, w tym błędnego działania aplikacji, utraty zaufania użytkowników, a w skrajnych przypadkach – do całkowitej awarii systemu.

Szczególnie istotnym obszarem, w którym ryzyko naruszenia integralności danych jest wyraźnie zauważalne, jest współczesny proces wytwarzania i wdrażania oprogramowania, realizowany najczęściej w modelu CI/CD (Continuous Integration / Continuous Deployment). Model ten umożliwia automatyczne testowanie, integrowanie i wdrażanie zmian kodu, co przyspiesza rozwój aplikacji i zwiększa efektywność zespołów programistycznych. Kluczową zaletą CI/CD jest ograniczenie ręcznych interwencji, jednak brak odpowiednich zabezpieczeń w tym procesie może prowadzić do poważnych incydentów bezpieczeństwa.

Jednym z istotnych zagrożeń jest możliwość ingerencji osób nieuprawnionych w system CI/CD. W przypadku uzyskania dostępu, atakujący mogą wprowadzić złośliwy kod, który zostanie automatycznie wdrożony do środowiska produkcyjnego, bez uprzedniego wykrycia. Może to skutkować m.in. kradzieżą danych użytkowników, eskalacją uprawnień, a także uzyskaniem trwałego dostępu do systemu przez osoby nieautoryzowane.

Dodatkowym zagrożeniem jest brak odpowiedniej weryfikacji modyfikacji wprowadzanych do kodu źródłowego. W sytuacji, gdy nie są stosowane mechanizmy kontroli jakości kodu, istnieje ryzyko wdrożenia zmian zawierających błędy logiczne, poufne dane (np. dane logowania), czy też niezweryfikowane zależności zewnętrzne. Szczególnie niebezpieczne może być korzystanie z bibliotek pochodzących z nieoficjalnych źródeł, które mogą zawierać znane luki w zabezpieczeniach bądź złośliwe funkcje. W takich przypadkach zalecane jest wdrożenie narzędzi typu Dependency Check lub SBOM (Software Bill of Materials), które umożliwiają identyfikację i analizę komponentów używanych w aplikacji.

Z punktu widzenia integralności danych istotnym aspektem jest również zabezpieczanie procesu serializacji i transmisji danych. Serializacja, czyli przekształcenie danych do formatu możliwego do zapisu lub transmisji, stanowi potencjalny wektor ataku, zwłaszcza gdy dane te są przesyłane bez odpowiednich mechanizmów weryfikacji. W przypadku braku zabezpieczeń, takich jak podpis cyfrowy czy mechanizmy kontroli integralności, możliwe jest przeprowadzenie ataku typu „man-in-the-middle” i manipulacja przesyłanymi danymi. W konsekwencji, aplikacja może zaakceptować zmodyfikowane dane jako

autentyczne, co może prowadzić do eskalacji uprawnień, naruszenia poufności lub pełnego przejęcia systemu.

W celu ograniczenia tego typu zagrożeń rekomenduje się stosowanie cyfrowych podpisów, szyfrowania transmisji danych oraz kontroli ich integralności przy użyciu sprawdzonych metod kryptograficznych. Dodatkowo, istotne jest zabezpieczenie całego łańcucha dostarczania oprogramowania — w tym systemów CI/CD — poprzez wdrożenie mechanizmów kontroli dostępu, dwuetapowej autoryzacji, przeglądu kodu oraz śledzenia zmian konfiguracyjnych. Kluczowe znaczenie ma także regularne skanowanie zależności oraz prowadzenie audytów bezpieczeństwa, które pozwalają wykryć potencjalne luki zanim zostaną wykorzystane przez osoby nieuprawnione.

9. A09 Security Logging and Monitoring Failures (Błędy logowania i monitorowania bezpieczeństwa)

Jednym z kluczowych elementów zapewnienia bezpieczeństwa w aplikacjach jest logowanie zdarzeń oraz monitorowanie systemu. Oznacza to, że aplikacja powinna rejestrować wszystkie istotne operacje zachodzące w jej obrębie, takie jak próby logowania, zmiany haseł, realizacja transakcji czy działania użytkowników mogące wskazywać na podejrzaną aktywność. Zarejestrowane dane podlegają następnie analizie oraz bieżącemu monitorowaniu, co umożliwia szybkie wykrycie potencjalnych zagrożeń i podjęcie odpowiednich działań zapobiegawczych. Skuteczne logowanie i monitorowanie odgrywają istotną rolę w reagowaniu na incydenty bezpieczeństwa oraz stanowi podstawę do ich późniejszej analizy, niezbędnej w kontekście audytów oraz dochodzeń powłamaniowych.

Problemy pojawiają się wówczas, gdy system nie rejestruje odpowiednich informacji lub brakuje mechanizmów monitorujących. Przykładami takich sytuacji są brak zapisów dotyczących nieudanych prób logowania, brak logów dotyczących istotnych operacji (np. modyfikacji konta użytkownika) lub przechowywanie logów wyłącznie lokalnie, bez odpowiednich mechanizmów tworzenia kopii zapasowych. Dodatkowym zagrożeniem jest generowanie logów nieczytelnych lub zawierających niewystarczające dane, co znacznie utrudnia analizę przyczyn incydentu. Równie niebezpieczny jest brak systemów generujących alerty w czasie rzeczywistym, które mogłyby informować administratorów o nietypowej aktywności, takiej jak próby włamań, nieautoryzowane operacje czy anomalie w ruchu sieciowym.

Jednym z najważniejszych aspektów logowania jest tworzenie tzw. śladu audytowego, który rejestruje historię operacji powiązanych z danymi i transakcjami w systemie. Zawiera on informacje dotyczące tego, kto, kiedy i jaką operację wykonał, a także z jakimi danymi była ona związana. Ślad audytowy powinien obejmować wszystkie kluczowe działania, takie jak zmiany w ustawieniach systemowych, logowania użytkowników, modyfikacje danych oraz transakcje o podwyższonym ryzyku, jak np. operacje finansowe. Stanowi on nieodzowne narzędzie umożliwiające późniejszą analizę i identyfikację odpowiedzialnych za ewentualne błędy lub działania niezgodne z przyjętymi procedurami bezpieczeństwa.

Integralność śladu audytowego ma zasadnicze znaczenie - dane w nim zawarte nie mogą być edytowane ani usuwane, gdyż mogłoby to prowadzić do zatarcia dowodów nieautoryzowanych działań. Z tego względu należy zabezpieczyć te informacje przy pomocy odpowiednich mechanizmów, takich jak podpisy cyfrowe czy technologie zapewniające nienaruszalność zapisów, w tym rozwiązania oparte na blockchainie.

W celu ograniczenia ryzyka związanego z niewystarczającym logowaniem i monitoringiem, aplikacje powinny rejestrować wszystkie próby logowania, błędy dostępu oraz inne istotne działania użytkowników, wraz z kontekstem umożliwiającym identyfikację potencjalnie niebezpiecznych kont. Logi powinny być zapisywane w formacie umożliwiającym ich późniejszą analizę przy użyciu dedykowanych systemów zarządzania logami, takich jak Splunk czy ELK Stack. Należy również zadbać o ich odpowiednie kodowanie, w celu zapobieżenia atakom wymierzonym w system logowania (np. atakom polegającym na wstrzykiwaniu złośliwego kodu). Kluczowe jest także przechowywanie kopii zapasowych logów, aby nie polegać wyłącznie na zapisach lokalnych, co chroni przed ich utratą w przypadku awarii.

Wszystkie operacje o wysokiej wartości, takie jak transakcje finansowe czy modyfikacje danych osobowych, powinny być objęte monitoringiem w ramach śladu audytowego, z odpowiednią ochroną przed manipulacją, tak aby nie było możliwości ich edycji lub usunięcia. Dodatkowo zaleca się wdrożenie systemów powiadamiania administratorów o

wykrytych zagrożeniach, umożliwiających natychmiastowe reagowanie na sytuacje takie jak próby włamań, nieautoryzowane zmiany w systemie czy podejrzane działania użytkowników.

10. A10 Server-Side Request Forgery (SSRF) (Fałszowanie żądań po stronie serwera)

Server Side Request Forgery (SSRF) stanowi podatność w aplikacjach internetowych, umożliwiającą atakującemu nakłonienie serwera do wysyłania żądań do nieautoryzowanych zasobów, często znajdujących się w wewnętrznych sieciach lub chronionych przez zapory sieciowe, sieci VPN bądź inne mechanizmy kontroli dostępu, takie jak listy ACL. W efekcie aplikacja, działając w kontekście zaufanego środowiska, realizuje żądanie do adresu URL wskazanego przez atakującego, a odpowiedź z takiego żądania może zostać zwrócona użytkownikowi. Tego typu podatność pozwala obejść zabezpieczenia sieciowe i uzyskać dostęp do wewnętrznych zasobów systemu.

Jednym z typowych przypadków ataku SSRF jest wykorzystanie adresów lokalnych, takich jak 127.0.0.1 lub localhost, albo prywatnych adresów IP, na przykład 192.168.1.8, celem uzyskania dostępu do zasobów serwera, które nie są bezpośrednio dostępne z zewnątrz. Przykładowo, żądanie typu http://192.168.1.8/filename może pozwolić na odczyt danych dostępnych jedynie wewnątrz infrastruktury. Innym scenariuszem jest częściowe SSRF, polegające na odczytywaniu danych z lokalnych plików systemowych, jak w przypadku żądania file:///etc/hosts, które może ujawnić szczegóły konfiguracji systemu operacyjnego. W przypadku blind SSRF atakujący nie otrzymuje bezpośredniej odpowiedzi z serwera, jednak może wpływać na adresy IP i porty, na które aplikacja wysyła zapytania. Takie podejście umożliwia na przykład testowanie dostępności wewnętrznych usług lub komunikację z zasobami kontrolowanymi przez atakującego, nawet przy braku widocznych danych zwrotnych.

W praktyce atak SSRF może zostać przeprowadzony poprzez funkcjonalności pozwalające użytkownikowi na wprowadzanie lub edycję adresów URL, jak np. w formularzach edycji profilu, gdzie złośliwie spreparowany URL może skłonić aplikację do wysłania żądania do serwera kontrolowanego przez atakującego. Może to skutkować ujawnieniem danych, dostępem do wewnętrznych systemów lub nawet uruchomieniem złośliwego oprogramowania.

Ochrona przed atakami SSRF wymaga wdrożenia szeregu mechanizmów zapobiegawczych. Należy przede wszystkim zadbać o właściwą walidację i oczyszczanie wszystkich danych wejściowych dostarczanych przez użytkownika, aby wykluczyć możliwość przekazania złośliwych adresów URL. Kluczowe znaczenie ma również stosowanie tzw. pozytywnej listy, obejmującej dozwolone adresy URL, porty i hosty docelowe, do których aplikacja może wysyłać zapytania. Istotnym elementem zabezpieczeń jest także dezaktywacja przekierowań HTTP, które mogłyby zostać wykorzystane do manipulowania ścieżką żądania. Dodatkowo, dostęp do usług wewnętrznych, takich jak Redis, MongoDB, Elasticsearch czy Memcached, powinien być zabezpieczony poprzez odpowiednie mechanizmy uwierzytelniania, co uniemożliwia nieautoryzowany dostęp. Wdrożenie odpowiednio skonfigurowanych zapór sieciowych, zdolnych do filtrowania ruchu wychodzącego i blokowania podejrzanych żądań, stanowi kolejne istotne ogniwo ochrony przed omawianym typem ataku.

4. WYNIKI

Testy z użyciem SQLmap wykazały obecność podatności typu SQL Injection w parametrze zapytania /rest/products/search?q=, co umożliwiło pobranie i analizę całej struktury bazy danych, w tym danych użytkowników oraz hashy haseł, z których część udało się złamać. Nmap ujawnił otwarte porty i aktywne usługi, takie jak FTP, HTTP, HTTPS czy proxy, co pozwoliło na ocenę powierzchni ataku i ukierunkowanie dalszych testów.

Przeprowadzone scenariusze Broken Access Control potwierdziły, że możliwe jest wykonywanie operacji w imieniu innych użytkowników przez manipulację parametrami żądań, a testy Cryptographic Failures ujawniły błędy w stosowaniu mechanizmów kryptograficznych, takie jak traktowanie Base64 jako zabezpieczenia czy stosowanie słabych algorytmów.

W obszarze Injection, poza klasycznym SQL Injection, wykryto także podatność XSS, umożliwiającą wstrzyknięcie i wykonanie złośliwego kodu JavaScript w przeglądarce użytkownika. Analiza pod kątem Insecure Design wykazała, że nieprawidłowe zaprojektowanie dostępu do katalogów pozwala na nieautoryzowane pobieranie poufnych dokumentów. W kategorii Security Misconfiguration

zidentyfikowano m.in. wyświetlanie szczegółowych stack trace'ów oraz podatność pozwalającą na podmianę grafik na stronie poprzez mechanizm directory traversal i redirect, co umożliwiło wstawienie zewnętrznego obrazka kota na pudełkach. Wykorzystując technikę null byte injection, udało się pobrać plik package.json.bak i przeanalizować listę zależności aplikacji. Narzędzia takie jak Snyk umożliwiły szybkie wykrycie podatnych bibliotek, które mogą być punktem wyjścia do dalszych ataków. Testy błędów identyfikacji i uwierzytelniania wykazały brak ochrony przed atakami brute force oraz możliwość wykorzystania domyślnych poświadczeń do przejęcia konta administratora, co stanowi krytyczną lukę bezpieczeństwa.

5. PODSUMOWANIE

5.1 Dyskusja

Analiza bezpieczeństwa OWASP Juice Shop wykazała, że nawet środowisko edukacyjne może wiernie odzwierciedlać wyzwania i zagrożenia obecne w rzeczywistych aplikacjach webowych. Przeprowadzone testy pozwoliły na praktyczne wykorzystanie i zrozumienie mechanizmów większości kluczowych podatności OWASP Top 10, takich jak SQL Injection, Broken Access Control, XSS, błędy kryptograficzne czy podatności wynikające z błędnej konfiguracji. Udało się wykazać, że skuteczne wykorzystanie narzędzi takich jak SQLmap, Burp Suite czy Nmap, w połączeniu z analizą architektury aplikacji i świadomym podejściem do testowania, pozwala nie tylko na wykrycie, ale i pełną eksploatację luk bezpieczeństwa.

Wyniki testów potwierdziły, że podatności mogą występować zarówno na poziomie kodu aplikacji, jak i w wykorzystywanych komponentach zewnętrznych czy konfiguracji środowiska. Szczególnie istotne okazały się luki w kontroli dostępu, błędy w implementacji mechanizmów kryptograficznych oraz możliwość wykorzystania przestarzałych lub podatnych bibliotek. Równie ważne były podatności wynikające z niewłaściwej konfiguracji aplikacji, które umożliwiły m.in. podmianę grafik na stronie czy uzyskanie dostępu do poufnych plików. Przeprowadzone scenariusze pokazały, że nawet pojedyncze zaniedbanie może prowadzić do poważnych konsekwencji, takich jak przejęcie konta administratora, wyciek danych czy kompromitacja całego systemu.

5.2 Wnioski

Badania potwierdziły, że regularne testowanie bezpieczeństwa aplikacji webowych, właściwa konfiguracja środowiska oraz systematyczna analiza wykorzystywanych zależności są kluczowe dla zapewnienia ochrony przed najczęściej występującymi zagrożeniami. Praktyczne wykorzystanie narzędzi do automatycznego wykrywania podatności w połączeniu z manualną analizą pozwala na szybkie wykrycie i eliminację luk, zanim zostaną wykorzystane przez osoby niepowołane. Wyniki pokazują, że nawet poprawnie zaprojektowane aplikacje mogą być narażone na ataki, jeśli nie są regularnie aktualizowane, monitorowane i testowane. Kluczowe znaczenie ma również edukacja zespołów deweloperskich oraz wdrażanie najlepszych praktyk w zakresie zarządzania bezpieczeństwem na każdym etapie cyklu życia oprogramowania.

5.3 Kierunki dalszych badań

W kontekście dalszych kierunków rozwoju kompetencji oraz badań nad bezpieczeństwem aplikacji webowych, istotnym krokiem może być rezygnacja z dalszego wykorzystywania OWASP Juice Shop jako głównego środowiska testowego. Mimo że aplikacja ta stanowi bardzo wartościowe narzędzie dydaktyczne i skutecznie symuluje wiele realnych zagrożeń zgodnych z klasyfikacją OWASP Top 10, jej charakter jako aplikacji demonstracyjnej sprawia, że posiada ona celowo wprowadzone podatności, które nie zawsze odzwierciedlają złożoność i nieprzewidywalność rzeczywistych środowisk produkcyjnych.

Z tego względu, naturalnym i logicznym krokiem w rozwoju może być skierowanie działań badawczych i testowych na bardziej realistyczne cele, takie jak uczestnictwo w programach typu bug bounty. W odróżnieniu od środowisk kontrolowanych, platformy takie jak HackerOne, Bugcrowd czy Intigriti umożliwiają testowanie aplikacji rzeczywistych firm. Często dużych, globalnych organizacji, które udostępniają swoje systemy do testów w zamian za zgłaszanie odnalezionych luk bezpieczeństwa. Uczestnictwo w takich programach pozwala nie tylko na praktyczne sprawdzenie nabytych umiejętności, ale również na konfrontację z systemami, które wykorzystują nowoczesne mechanizmy obronne, posiadają skomplikowaną logikę biznesową i różnorodne integracje technologiczne.

Przejście od środowiska edukacyjnego do programów bug bounty to również zmiana podejścia metodologicznego, wymaga ono większej precyzji, cierpliwości oraz odpowiedzialności, zarówno prawnej, jak i etycznej. Testy prowadzone w takich warunkach muszą być zgodne z zasadami określonymi przez właścicieli aplikacji, a każde zgłoszenie podatności powinno być odpowiednio udokumentowane i zweryfikowane, co stanowi dodatkowy

element rozwoju kompetencji zawodowych w obszarze cyberbezpieczeństwa.

Literatura

1. OWASP Juice Shop – oficjalna dokumentacja projektu https://owasp.org/www-project-juice-shop/ (dostęp: 19.05.2025)
2. The OWASP Top 10 – oficjalny dokument https://owasp.org/www-project-top-ten/ (dostęp: 19.05.2025)
3. SQLmap – dokumentacja narzędzia https://sqlmap.org/ (dostęp: 19.05.2025)
4. Nmap Documentation – oficjalny przewodnik https://nmap.org/docs.html (dostęp: 19.05.2025)
5. Burp Suite Documentation – instrukcja użytkowania https://portswigger.net/burp/documentation (dostęp: 19.05.2025)
6. GitHub – repozytorium OWASP Juice Shop https://github.com/juice-shop/juice-shop (dostęp: 19.05.2025)