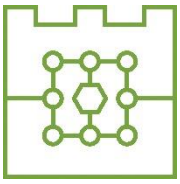
**Politechnika Krakowska**
**im. Tadeusza Kościuszki**

Wydział Informatyki i Telekomunikacji



**Damian Skarbiecki**

Numer albumu: 131829

# Analiza i ocena skuteczności narzędzi do skanowania podatności w aplikacjach webowych

# Analysis and Evaluation of the Effectiveness of Vulnerability Scanning Tools for Web Applications

**Praca magisterska**
**na kierunku Informatyka**
**specjalność Cyberbezpieczeństwo**

Praca wykonana pod kierunkiem:
**Dr Radosław Kycia**

**Kraków, 2025**

## Streszczenie

W pracy porównano trzy narzędzia do automatycznego skanowania podatności w aplikacjach webowych, a mianowicie OWSAP ZAP, Burp Suite oraz Smart Scanner. Badanie sprawdza ich zdolność oraz efektywność w wykrywaniu podatności, a także dokładność generowanych raportów. Wyniki pokazują, że OWSAP ZAP charakteryzuje się duża skutecznością w wykrywaniu różnorodnych podatności. Dostarcza on również szczegółowy raport z testów. Narzędzie Burp Suite również efektywnie identyfikuje liczne zagrożenia, jednak jego raporty nie są tak szczegółowe jak te generowane przez OWSAP ZAP. Smart Scanner za to wykazał się najmniejszą efektywnością wykrywania luk mimo że czas skanów był najkrótszy. Jego raporty są również najmniej czytelne. Analiza pokazuje znaczenie wyboru odpowiedniego narzędzia w zależności od potrzeb i priorytetów zespołu testującego.

## Abstract

This study compares the effectiveness of three automated web application security testing tools: OWSAP ZAP, Burp Suite, and Smart Scanner. The evaluation is focused on their efficiency in indentifying vulnerabilities and the comprehensiveness of their generated reports. The findings indicate that OWSAP ZAP demonstrates high effectiveness in detecting various vulnerabilities while providing detailed reports. Burp Suite also effectively identifies numerous threats, however, its reports are less detailed compared to OWSAP ZAP. Smart Scanner offers the fastest scanning time but detects the fewest vulnerabilities, and its reports are the least comprehensive. The analysis emphasizes the importance of selecting the appropriate tool based on the specific needs and priorities of the testing team.

## Słowa kluczowe:

Skanowanie podatności, Bezpieczeństwo aplikacji internetowych, Testy penetracyjne, Narzędzia do testowania bezpieczeństwa, Wstrzykiwanie kodu SQL

## Keywords:

Vulnerability scanning, Web application security, Penetration testing, Security testing tools, SQL Injeciton

# Spis treści

# 1. Wprowadzenie

## *1.1 Cel pracy*

Głównym celem pracy jest przeprowadzenie kompleksowej analizy oraz oceny skuteczności narzędzi do skanowania podatności w aplikacjach webowych. Badanie skupi się na identyfikacji kluczowych czynników wpływających na efektywność tych narzędzi, porównaniu różnych rozwiązań dostępnych na rynku pod kątem ich zdolności do wykrywania i zarządzania zidentyfikowanymi podatnościami. Dodatkowo, praca będzie miała na celu zaproponowanie praktycznych wskazówek i sugestii dotyczących optymalnego wykorzystania narzędzi skanujących w procesie zabezpieczania aplikacji webowych.

## *1.2 Zakres pracy*

Praca skupia się na trzech narzędziach: OWSAP ZAP, Burp Suite, Smart Scanner, które służą do wykrywania podatności w aplikacjach webowych. W ramach testów porównano wykrywanie i raportowanie konkretnych wektorów ataku: wstrzykiwanie kodu SQL, wstrzykiwanie skryptu do stron internetowych, szyfrowanie komunikacji i obchodzenie ścieżek.

# Część Teoretyczna

### *1.3 Kontekst problematyki bezpieczeństwa aplikacji webowych*

Bezpieczeństwo aplikacji webowych w dzisiejszym świecie IT to prawdziwe wyzwanie. Zaawansowany rozwój technologii oraz swobodny dostęp do internetu sprawiają, że aplikacje te odgrywają centralną rolę w funkcjonowaniu firm i organizacji. Pozwalają one na zarządzanie danymi, komunikację z klientami i świadczenie usług internetowych. Mimo to ich globalna dostępność, a także częste interakcje z niezweryfikowanymi użytkownikami mogą prowadzić do licznych zagrożeń bezpieczeństwa.

Rosnąca liczba cyberataków na aplikacje internetowe ujawnia mnóstwo podatności wynikających z nieadekwatnej ochrony danych, błędów w kodzie źródłowym a także błędów konfiguracyjnych. Według raportu opublikowanego przez OWSAP (Open Web Application Security Project), aplikacje webowe często zawierają krytyczne luki w zabezpieczeniach, które mogą prowadzić do wycieku danych, nieuprawnionego dostępu a także przejęcia kontroli na całym systemem. Najczęstsze podatności to na przykład wstrzykiwanie kodu (z ang, SQL Injection), wstrzykiwanie skryptów do stron internetowych (z ang. Cross-Site Scripting), czy nieodpowiednie zarządzanie sesjami lub brak kontroli dostępu.

Bezpieczeństwo aplikacji webowych dotyczy nie tylko aspektów technicznych, ale także kwestii prawnych i biznesowych. Wyciek informacji może wyrządzić organizacji znaczące straty pieniężne oraz prowadzi do utraty reputacji, co może znacząco wpłynąć na jej działalność. Na przykład wprowadzenie ogólnego rozporządzenia o ochronie danych (RODO/GDPR)[15] wymusiło na europejskich firmach dodatkowe obowiązki dotyczące zapewnienia ochrony prywatności użytkowników.

W raz z rozwojem technologii, migracja aplikacji do chmury obliczeniowej staje się normą, co powoduje dużo wyzwań związanych z bezpieczeństwem. Aplikacje wymagają dostosowania sposobu, w jaki są zabezpieczane. Współczesne aplikacje często wykorzystują skomplikowane infrastruktury wielowarstwowe, mikroserwisy, API itp. Ze względu na to zwiększa się pula celów podatnych na ataki i wymaga bardziej nowoczesnych i sprytnych narzędzi oraz metod lokalizowania podatności.

Zrozumienie kontekstu problematyki bezpieczeństwa aplikacji webowych pozwala na odkrycie znaczenia analizy i weryfikacji tych aplikacji pod kątem luk w zabezpieczeniach. Prowadzenie regularnych testów bezpieczeństwa, implementacja odpowiednich

mechanizmów ochronnych oraz stosowanie narzędzi do skanowania podatności staje się bezwzględnie potrzebnym elementem zagwarantowania bezpieczeństwa aplikacji webowych i zminimalizowania ryzyka związanego z atakami cybernetycznymi.

### *1.4 Znaczenie skanowania podatności w procesie zabezpieczania aplikacji*

Skanowanie podatności w procesie zabezpieczania aplikacji internetowych jest znaczącym problemem. Pozwala na wczesne wykrycie i identyfikację lub bezpieczeństwa w oprogramowaniu, które mogłoby zostać wykorzystane do przeprowadzenia cyberataków. Metoda ta polega na automatycznym lub manualnym skanowaniu kodu aplikacji, konfiguracji oraz struktury pod kątem popularnych podatności jak wstrzykiwanie kodu, wstrzykiwanie skryptu do stron internetowych, czy niepoprawnego zarządzania sesjami użytkowników. Regularność skanowania pomaga organizacjom proaktywnie zarządzać ryzykiem, by zminimalizować liczbę krytycznych luk w aplikacji, a także zwiększać bezpieczeństwo swojego oprogramowania.

Podstawowe znaczenie skanowania podatności wynika z kilku istotnych aspektów:

- **Identyfikacja Znanych Zagrożeń:** Skanery podatności automatycznie wykrywają powszechnie znane luki bezpieczeństwa opisane w bazach podatności takich jak CVE (popularne podatności i luki, z ang. Common Vulnerabilities and Exposures) lub OWASP Top 10. Znalezienie takich podatności w kodzie źródłowym lub konfiguracji pozwala na natychmiastowe podjęcie działań zabezpieczających[1, 17].

- **Ocena Ryzyka:** Raporty generowane przez narzędzia skanujące, pozwalają administratorom na przypisanie każdej podatności określony poziom ryzyka. Pozwala to na priorytetyzację działań naprawczych, skupiając się najpierw na eliminacji najbardziej krytycznych zagrożeń, które mogą mieć największy wpływ na bezpieczeństwo aplikacji.

- **Oszczędność Czasu i Zasobów:** Automatyzacja procesu skanowania znacząco redukuje czas potrzebny na manualne przeszukiwanie kodu i identyfikację potencjalnych luk. Z racji tego narzędzia do wykrywania podatności są konieczne w dużych projektach, gdzie liczba linii kodu czy zastosowanych technologii może być bardzo duża.

- **Spełnienie Wymagań Regulacyjnych i Standardów:** W wielu branżach, takich jak finanse, ochrona zdrowia czy sektor publiczny, istnieją regulacje dotyczące

bezpieczeństwa danych, które nakładają obowiązek przeprowadzania regularnych testów bezpieczeństwa. Skanowanie podatności pozwala na spełnienie tych wymagań i zaprezentowaniu zgodności z obowiązującymi przepisami (np. PCI DSS, RODO/GDPR)[15].

- **Minimalizacja Ryzyka Ataków:** Regularne skanowanie pozwala na wczesne wykrywanie luk, zanim zostaną one wykorzystane przez atakujących. Dzięki temu organizacje mogą reagować proaktywnie, usuwając podatności przed ich potencjalnym wykorzystaniem w rzeczywistych atakach.

- **Wspieranie Procesu Rozwoju Aplikacji (DevSecOps):** W kontekście nowoczesnych metodologii tworzenia oprogramowania, takich jak DevSecOps, skanowanie podatności jest wbudowane w cykl życia aplikacji. Narzędzia te mogą być zintegrowane z systemami ciągłej integracji (CI/CD – proces, który polega na zautomatyzowaniu kroków tworzenia oprogramowania składający się z ciągłej integracji CI oraz ciągłego dostarczania CD), umożliwiając automatyczną kontrolę bezpieczeństwa w każdej fazie rozwoju – od etapu pisania kodu po wdrożenie produkcyjne.

Podsumowując, skanowanie podatności jest bardzo powszechnym elementem zarządzania bezpieczeństwem aplikacji sieciowych. Pomaga zidentyfikować aspekty bezpieczeństwa, ocenić ryzyko oraz umożliwia efektywne reagowanie na potencjalne zagrożenia. Dzięki temu firmy mogą skutecznie zabezpieczać swoje zasoby przed coraz bardziej zaawansowanymi i złożonymi atakami, jednocześnie optymalizując procesy związane z bezpieczeństwem oraz minimalizując ryzyka związane z cyberprzestępczością. Dlatego w dzisiejszych czasach oprogramowanie skanujące podatności odgrywa ważną rolę w zabezpieczaniu aplikacji.

### *1.5 Przegląd literatury dotyczącej bezpieczeństwa aplikacji webowych*

Koncepcja bezpieczeństwa aplikacji webowych to ważne i szeroko badane zagadnienie, a jednym z najbardziej uznawanych dokumentów patrzących na bieżące problemy bezpieczeństwa, to ranking monitorowany przez projekt OWASP Top Ten[1]. Jest to cyklicznie aktualizowany katalog dziesięciu najczęstszych i najniebezpieczniejszych podatności w aplikacjach webowych, stanowiący odniesienie w dziedzinie praktyk i badań tego obszaru. Lista OWASP Top Ten zawiera m.in. takie podatności jak wstrzykiwanie kodu

SQL, złamanej autentykacji, czy wstrzykiwanie skryptów do storn internetowych, które często są przedmiotem badań nad skutecznością narzędzi skanujących.

Literatura naukowa często odnosi się do tej listy jako standardu przy ocenie efektywności metod testowania aplikacji. W ostatnich latach przeprowadzono szereg badań mających na celu ocenę skuteczności narzędzi do skanowania podatności w aplikacjach webowych. Istotnym punktem odniesienia w tej dziedzinie jest artykuł pt. „*Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners"[2]*, w którym dokonano przeglądu najpopularniejszych narzędzi skanujących oraz ich zdolności do wykrywania najczęstszych podatności.

Wyniki analizy[2] pokazują, że choć skanery takie jak OWASP ZAP[4], Arachni[7] czy w3af[16] są bardzo skuteczne w detekcji popularnych luk bezpieczeństwa takich jak wstrzykiwanie kody SQL czy wstrzykiwanie skryptu do stron internetowych, mają jednak trudności z wykrywaniem bardziej skomplikowanych i zaawansowanych podatności związanych np. z logiką aplikacji lub zarządzaniem sesjami. Badanie pokazuje również, że w przypadku wielu narzędzi występuje częsty problem z fałszywymi alarmami. Oznacza to, że część wykrytych podatności jest błędna, niemająca rzeczywistego wpływu na bezpieczeństwo aplikacji.

Autorzy artykułu podkreślają także potrzebę systematycznego testowania skuteczności narzędzi w różnych środowiskach i scenariuszach, aby lepiej zrozumieć ich ograniczenia. Efektywność skanerów jest często zależna od ich konfiguracji oraz charakterystyki samej aplikacji. Na przykład struktury bazy danych, sposobu przesyłania danych czy mechanizmów autoryzacji. Autorzy zaznaczają również problemy z wykrywaniem podatności takich jak zła konfiguracja bezpieczeństwa (a ang. Security Misconfiguration) czy wykrywanie wrażliwych danych (z ang. Sensitive Data Exposure), ponieważ wymagają bardziej zaawansowanej analizy. Aby zmniejszyć liczbę przeoczonych luk i zwiększyć skuteczność ważne jest integrowanie skanerów z innymi narzędziami bezpieczeństwa. Podsumowując, chociaż istniejące narzędzia są w stanie efektywnie wspierać proces zabezpieczania aplikacji webowych, to ich pełne wykorzystanie wymaga znajomości ich ograniczeń oraz potrzeb samej aplikacji[2].

### *1.6 Dostępne narzędzia do skanowania podatności*

W dziedzinie bezpieczeństwa aplikacji internetowych istnieje wiele różnych narzędzi, pozwalających na detekcję podatności w aplikacjach. Skanery podatności są koniecznym narzędziem w procesie audytu bezpieczeństwa, testach penetracyjnych a także analizie zabezpieczeń. Narzędzia te mogą automatycznie identyfikować potencjalne, najpopularniejsze luki w zabezpieczeniach, takie jak SQL wstrzykiwanie kodu SQL, wstrzykiwanie skryptu do stron internetowych, czy błędy związane z konfiguracją. Poniżej przedstawiono kilka najważniejszych narzędzi dostępnych na rynku, z podziałem na kategorie otwarto źródłowe (z ang. Open-source) oraz komercyjne.

**Otwarto źródłowe:**

- **OWASP ZAP**

Jest to jeden z najpopularniejszych skanerów podatności typu otwarto źródłowego, opracowany i rozwijany przez OWASP. OWASP ZAP jest narzędziem typu pośredniczącego, które może przechwytywać i modyfikować ruch sieciowy między przeglądarką a aplikacją. Oferuje on duży wybór narzędzi do wykrywania luk, takich jak wstrzykiwanie kodu SQL, wstrzykiwanie skryptu do stron internetowych, oraz innych podatności z rankingu OWASP Top Ten[1]. OWSAP ZAP oferuje również intuicyjny interfejs użytkownika oraz możliwość integracji z systemami CI/CD[4].

**Zalety:** Bezpłatny, elastyczny oraz bardzo dobrze udokumentowany, aktywnie rozwijany przez społeczność.

**Wady:** Mniej efektywny w wykrywaniu zaawansowanych luk, wymaga skomplikowanej konfiguracji w bardziej złożonych aplikacjach.

- **w3af (Web Application Attack and Audit Framework)**

Jest wysoce rozwiniętym narzędziem, które jest używane do zarówno ataków jak i testów bezpieczeństwa aplikacji webowych. Oferuje ono bogaty zestaw wtyczek pozwalających na wykrywanie różnego rodzaju podatności, takich jak wstrzykiwanie kodu SQL, wstrzykiwanie skryptu do stron internetowych, zmuszenie przeglądarki do wykonania złośliwej akcji (z ang. Cross-Site Request Forgery). Użytkownicy mogą dostosowywać

skanowania do swoich indywidualnych potrzeb poprzez zastosowanie modularnej struktury narzędzia[16].

**Zalety:** Szerokie możliwości konfiguracji, duży zestaw wtyczek, dostępność raportów w formacie XML i JSON.

**Wady:** Mniejsza społeczność i wsparcie w porównaniu do OWASP ZAP, interfejs linii komend jest mniej intuicyjny dla nowych użytkowników.

- **Arachni**

Jest to rodzaj narzędzia oferujący automatyczne wykrywanie różnorodnych podatności aplikacji internetowych. Na przykład luk związanych z zarządzaniem sesjami, uwierzytelnianiem oraz błędami konfiguracyjnymi. Rozproszone skanowanie oferowane przez narzędzie jest szczególnie przydatne w przypadku większych aplikacji[7].

**Zalety:** Rozproszone skanowanie, wsparcie dla wielu formatów raportowania, wydajność w dużych aplikacjach.

**Wady:** Interfejs użytkownika jest mniej intuicyjny niż w przypadku OWSAP ZAP, wymaga większych zasobów systemowych do rozproszonego skanowania.

**Komercyjne:**

- **Burp Suite Professional**

Jest to jedno z najczęściej używanych narzędzi do testowania bezpieczeństwa aplikacji webowych przez profesjonalnych testerów penetracyjnych. Oferuje ono zaawansowane możliwości manualnych i automatycznych testów penetracyjnych, w tym wykrywanie podatności takich jak wstrzykiwanie kod SQL, wstrzykiwanie skryptu do stron internetowych, oraz luki związane z logowaniem i zarządzaniem sesjami. Posiada ono rozbudowany interfejs i moduły, które mogą być dostosowywane do specyficznych potrzeb[6].

**Zalety:** Zaawansowane funkcje testowania manualnego, duże możliwości konfiguracyjne, wsparcie techniczne i dokumentacja.

**Wady:** Wysoki koszt licencji, skomplikowany dla początkujących użytkowników.

- **Acunetix**

Jest to komercyjny skaner podatności oferujący szybkie oraz zautomatyzowane testy aplikacji webowych. Oprogramowanie to jest szczególnie skuteczne w wykrywaniu luk z listy OWASP Top Ten oraz błędów związanych z uwierzytelnianiem i zarządzaniem sesjami. Dzięki integracji z narzędziami CI/CD, Acunetix może być używany w procesach DevSecOps do ciągłego monitorowania bezpieczeństwa aplikacji[12].

**Zalety:** Szybkość skanowania, intuicyjny interfejs, zaawansowane raportowanie i wsparcie techniczne.

**Wady:** Wysoki koszt, ograniczony w darmowej wersji próbnej.

Podsumowując, narzędzia otwarto źródłowe służą mniejszym firmą i organizacjom, które nie mogą sobie pozwolić na specjalistyczne drogie narzędzia. Otwarty kod źródłowy pozwala na dostosowanie narzędzi do indywidualnych potrzeb oraz integracji z własnymi systemami. Jednym z kolejnych plusów jest duża liczba użytkowników społeczności. Narzędzia komercyjne jednak oferują bardziej zaawansowane funkcję oraz wsparcie techniczne czy szybsze aktualizacje związane z nowymi zagrożeniami. Mimo że ich cena nie należy do niskich to narzędzia te oferują między innymi kompleksowe rozwiązania do zarządzania bezpieczeństwem dużych aplikacji, gdzie narzędzia otwarto źródłowe mogą być mniej skuteczne przy bardziej zaawansowanych systemach. Wybór odpowiedniego narzędzia zależy od specyficznych potrzebach organizacji takich jak budżet, poziom technicznego wsparcia, oraz typ aplikacji, które mają być testowane. Narzędzia otwarto źródłowe mogą sprawdzać się lepiej dla mniejszych zespołów, które dysponują odpowiednimi umiejętnościami technicznymi. Natomiast narzędzia komercyjne, pomimo iż są droższe, to oferują wyższy poziom automatyzacji i wsparcia, co czyni je atrakcyjnymi dla dużych przedsiębiorstw[3].

### *1.7 Analiza literatury dotyczącej metodologii skanowania podatności*

Skanowanie aplikacji webowych pod względem ich podatności na ataki jest kluczowym elementem zapewniania ich bezpieczeństwa. Metodologie skanowania podatności ewoluowały wraz z postępem technologicznym oraz wzrostem liczby i złożoności zagrożeń. W literaturze omawiane są różne podejścia oraz techniki wykorzystywane do skanowania

aplikacji webowych w celu identyfikacji luk bezpieczeństwa. Przykładami są skanowanie ręczne, automatyczne i hybrydowe.

- **Skanowanie ręczne:** Polega na ręcznym przeszukaniu kodu oraz architektury aplikacji przez pentesterów oraz specjalistów ds. Bezpieczeństwa w celu zidentyfikowania luk. Proces ten może być czasochłonny jednak jest niezbędny w sytuacjach, gdy złożone błędy są trudne do wykrycia przez automatyczne narzędzia. W badaniach podkreślane jest, że skanowanie ręczne pozwana na większą elastyczność w analizie specyficznych zagrożeń, co pozwala uzyskać dokładniejsze wyniki. Przykładami takiej techniki są analiza kodu źródłowego czy testowanie wejścia aplikacji pod kątem wstrzykiwania danych[5].
- **Skanowanie automatyczne:** To technika polegająca na używaniu zautomatyzowanych narzędzi takich jak OWSAP ZAP[4] czy Arachni[7], które służą do szybkiego wykrywania znanych podatności. Analizują one aplikację w oparciu o wcześniej zdefiniowane wzorce. Narzędzia te korzystają z bazy sygnatur podatności co pomaga im w szybki sposób przeskanować duże aplikacje. Badania jednak wskazują, że automatyczne narzędzia często generują wiele fałszywych alarmów, przez co ocena rzeczywistego poziomu zagrożenia może być trudniejsza. Automatyczne skanowanie jest szybsze w porównaniu do ręcznego. Jednak jego skuteczność zależy od regularnych aktualizacji bazy sygnatur oraz złożoności aplikacji[5].
- **Skanowanie hybrydowe:** W literaturze często omawiane jest połączenie metod skanowania ręcznego z automatycznym. W takim przypadku narzędzia automatyczne wykonują podstawową analizę, która jest potem uzupełniana o ręczne testowanie w trudniejszych obszarach. Takie podejście minimalizuje liczbę fałszywych alarmów, gdyż wyniki uzyskane przez narzędzia mogą być szybko weryfikowane przez ekspertów[5].

Narzędzia do skanowania podatności aplikacji webowych korzystają z wielu różnych technik umożliwiających wykrywanie zarówno znanych, jak i nowych zagrożeń. Główne techniki obejmują skanowanie oparte na sygnaturach czyli skanowaniu aplikacji w poszukiwaniu wzorców znanych podatności. Te sygnatury opierają się na bazach danych, takich jak CVE[17] czy baza danych OWASP Top Ten[1]. Skanowanie oparte na sygnaturach skutecznie wykrywa znane zagrożenia, jednak ma problemy ze znalezieniem nowych lub specyficznych dla danej aplikacji luk. Przykładem narzędzia wykorzystującym tę technikę jest OpenVAS. Inną techniką jest tak zwane zamazywanie (z ang. Fuzzing), Działa on poprzez

wprowadzanie nieprawidłowych, losowych lub błędnie sformatowanych danych do aplikacji. Dzięki temu wywołuje on nieoczekiwane zachowania aplikacji. Celem tej metody jest ujawnienie podatności, które mogą być trudne do wykrycia tradycyjnymi metodami. Zamazywanie jest popularnym podejściem w narzędziach takich jak na przykład OWSAP ZAP czy Burp Suite. Inną techniką jest analizowanie statyczne, które analizuje kod źródłowy bez uruchamiania aplikacji. Jest to często stosowana technika na wczesnym etapie rozwoju oprogramowania, pomaga wykrywać potencjalne problemy na poziomie kodu, takie jak źle obsługiwane dane wejściowe, podatności na wstrzykiwanie kodu SQL czy wstrzykiwanie skryptu do stron internetowych. Narzędzia są takie jak SonarQube, które mogą oferować analizę statyczną w czasie rzeczywistym pomagając zespołowi deweloperskiemu identyfikować problemy, podczas pisania kodu. Na koniec, analizując techniki związane z testowaniem wydawania oprogramowania, nie sposób pominąć techniki dynamicznej, która w przeciwieństwie do analizy statycznej testuje aplikacje w czasie rzeczywistym, symulując ataki na działającą aplikację. W przeciwieństwie do analizy statycznej, technika ta pozwala wykryć podatności, które mogą być aktywne w środowisku produkcyjnym, w tym podatności na ataki typu wstrzykiwanie kodu SQL, ataki typu brutalnej siły (z ang. Brute Force) czy zmuszanie przeglądarki do wykonania złośliwej akcji. Narzędzia dynamiczne, takie jak OWSAP ZAP i Burp Suite, są powszechnie stosowane w tej metodzie[9].

# 2. Metodologia

## *2.1 Plan badawczy*

Plan badawczy jest kluczowym elementem zapewniającym systematyczność i rzetelność analizy narzędzi do skanowania podatności aplikacji webowych. Obejmuje wszystkie etapy, od wyboru narzędzi i środowiska testowego po szczegółowe zestawienie wyników. Dzięki odpowiedniej strukturze badanie umożliwia obiektywną ocenę efektywności narzędzi w różnorodnych scenariuszach.

Podstawowym celem badania jest weryfikacja, jak skutecznie narzędzia wykrywają różne rodzaje podatności, takie jak wstrzykiwanie kodu SQL, wstrzykiwanie skryptu do stron internetowych, czy brak kontroli dostępu. Ważnym aspektem jest też analiza użyteczności narzędzi oraz ich adaptacyjności do nowych rodzajów zagrożeń. Badanie pozwala także ocenić, które narzędzia są najbardziej przydatne w określonych kontekstach.

Badanie obejmuje:

- **Analizę dwóch aplikacji**:
  1. WebGoat – aplikacja testowa z zaimplementowanymi celowo podatnościami.
  2. Własna aplikacja zbudowana jako przykład realistycznych scenariuszy zagrożeń.
- **Testowanie różnych kategorii podatności**:
  1. Wstrzykiwanie SQL.
  2. Wstrzykiwanie skryptu do stron internetowych.
  3. Zmuszanie przeglądarki do wykonania złośliwej akcji.
  4. Brak odpowiedniego szyfrowania danych.

Proces badawczy:

- **Przygotowanie środowiska testowego**:
  1. Uruchomienie i konfiguracja obu aplikacji.
  2. Instalacja wybranych narzędzi: OWASP ZAP, Burp Suite i SmartScanner.
- **Wykonanie testów**:
  1. Każde narzędzie zostanie użyte do przeszukiwania obu aplikacji.
  2. Testy będą przeprowadzane w zdefiniowanych scenariuszach.

- **Rejestracja wyników**:
    1. Zapisywanie liczby wykrytych podatności.
    2. Dokumentacja czasu potrzebnego na przeprowadzenie skanów.

Badanie sukcesu narzędzi będzie oparte na:

- **Precyzji wykrywania** – liczba prawdziwych pozytywnych wyników.
- **Prostocie obsługi** – ocena złożoności interfejsu.
- **Szybkości działania** – czas wymagany na pełny skan.

Analiza powinna wskazać:

- Narzędzia najlepiej radzące sobie z określonymi typami podatności.
- Obszary, w których narzędzia wymagają ulepszeń.
- Rekomendacje dotyczące optymalnego wykorzystania narzędzi w praktyce.

Plan badawczy stanowi fundament analizy, który pozwoli na dokładne porównanie skuteczności narzędzi i ich przydatności w zabezpieczaniu aplikacji webowych. Zebrane wyniki mogą być także podstawą do wskazania kierunków dalszego rozwoju narzędzi do skanowania podatności.

## *2.2 Klasyfikacja podatności aplikacji webowych*

Bezpieczeństwo aplikacji webowych jest bardzo ważnym elementem współczesnych systemów informatycznych, zwłaszcza w obliczu coraz częstszych cyberataków. Skanowanie podatności aplikacji webowych opiera się na identyfikacji luk w zabezpieczeniach, które mogą być wykorzystane przez atakujących. W tym kontekście, zrozumienie klasyfikacji podatności jest fundamentem skutecznego zabezpieczania aplikacji. W tej części zostaną omówione najczęściej występujące podatności zgodnie z ich klasyfikacją i w odniesieniu do standardów, takich jak OWASP Top Ten[1].

**Wstrzykiwanie kodu SQL** - To jedna z najbardziej niebezpiecznych luk w systemach, pozwalająca atakującemu na manipulację zapytaniami SQL. Na przykład, gdy użytkownik wprowadzi jakieś dane, które są bezpośrednio wstawiane do zapytania SQL bez odpowiedniej walidacji. Może to umożliwić np. odkrycie zawartości tabeli z hasłami użytkowników. Wstrzykiwanie kodu SQL może mieć wiele form[10]:

- Błąd w zapytaniu – pozwala wyświetlać błędy bazy danych.
- Wstrzykiwanie kodu SQL oparte na wiązaniu – łączy wiele zapytań SQL w jedno.
- Ślepe wstrzykiwanie kodu SQL – brak bezpośrednich komunikatów zwrotnych, ale atakujący może na podstawie czasów odpowiedzi odgadnąć dane.

**Wstrzykiwanie skryptu do stron internetowych (XSS)** - jest to pojęcie polegające na wstrzykiwaniu kodu JavaScript w odpowiedzi serwera. XSS można także użyć do przejęcia kontroli nad sesją użytkownika, zmiany zwartości strony lub kradzieży danych. XSS występuje w trzech wariantach[8]:

- Odbity atak XSS – najczęściej spotykany wariant, w którym złośliwy kod JavaScript jest "odbijany" w odpowiedzi serwera po wprowadzeniu danych przez użytkownika. Kod ten może wykraść ciasteczka sesyjne lub przeprowadzić inne złośliwe operacje.
- Przechowany XSS – kod jest zapisany w bazie danych i wyświetlany dla wielu użytkowników, co czyni go bardziej niebezpiecznym. Przykładem może być forum internetowe, gdzie użytkownik wstawia złośliwy skrypt w komentarzach, które są później odtwarzane na stronie dla każdego odwiedzającego.
- Bazujący na dokumencie HTML XSS – polega na manipulacji dokumentem HTML bezpośrednio przez przeglądarkę, bez interakcji z serwerem.

**Zmuszanie przeglądarki do wykonania złośliwej akcji (CSRF)** - umożliwia atakującemu wykonanie złośliwego żądania w imieniu zalogowanego użytkownika. Atak jest szczególnie groźny w sytuacjach, gdy użytkownik jest zalogowany do banku lub portalu społecznościowego. Poprzez np. spreparowany link, atakujący może wysłać żądanie zmiany hasła lub wykonania przelewu. Kluczowe w ochronie przed CSRF jest stosowanie tokenów CSRF, które zabezpieczają formularze przed tego typu atakami[8].

**Niezabezpieczona deserializacja (z ang. Insecure Deserialization)** - podatność ta pojawia się, gdy dane pochodzące od użytkownika są deserializowane (np. dane JSON, XML), a aplikacja bez odpowiedniej walidacji dopuszcza manipulację. Atakujący może zmodyfikować struktury danych w taki sposób, by uzyskać nieautoryzowany dostęp do aplikacji lub wstrzyknąć złośliwy kod. Przykład to atak na deserializację obiektów w Javie, który może prowadzić do zdalnego wykonania kodu[8].

**Brak odpowiedniej kontroli dostępu** - niewystarczająca kontrola dostępu (z ang. Broken Access Control) to bardzo poważna luka. Może ona wystąpić, gdy aplikacja nie sprawdza, czy użytkownik ma odpowiednie uprawnienia do wykonania określonych operacji. Przykład: użytkownik z kontem "user" może uzyskać dostęp do zasobów dostępnych tylko dla administratorów, poprzez manipulację URL (np. wpisując /admin/ w adresie strony)[8].

**Bezpieczeństwo komunikacji** - aplikacje, które nie wykorzystują szyfrowania (np. HTTPS), narażają dane przesyłane pomiędzy przeglądarką użytkownika a serwerem na przechwycenie przez atakującego (np. ataki typu człowiek w środku, z ang. Man-in-the-Middle). Zabezpieczenie transmisji danych poprzez certyfikaty SSL/TLS jest kluczowym elementem ochrony danych wrażliwych, takich jak hasła, numery kart kredytowych itp[8].

**Bezpieczne uwierzytelnianie i zarządzanie sesjami** - niewłaściwe zarządzanie sesjami, takie jak brak odpowiednich zabezpieczeń w procesie logowania, pozwala atakującym na przejęcie sesji użytkownika. Przykład: ciasteczka sesyjne są przechowywane w niezabezpieczony sposób, co umożliwia ich kradzież i późniejsze wykorzystanie. Popularne ataki to przejęcie ciasteczka sesji poprzez atak XSS [10].

### *2.3 Wybór narzędzi do analizy skuteczności*

Do oceny skuteczności wykrywania podatności aplikacji webowych wybrano trzy narzędzia: Burp Suite, OWASP ZAP oraz Smart Scanner. Wybór opiera się na różnorodności ich funkcji, popularności w branży bezpieczeństwa, a także ich zdolności do obsługi OWASP Top Ten.

- **Burp Suite**: Profesjonalne narzędzie komercyjne z darmową wersją (Community Edition). Oferuje zaawansowane funkcje, takie jak przechwytywanie ruchu, skaner podatności oraz narzędzia do manualnej eksploracji aplikacji. Jego zaletą jest rozbudowany zestaw narzędzi do analizy dynamicznej i możliwość tworzenia własnych skryptów testowych.
- **OWASP ZAP**: Otwarto źródłowe narzędzie stworzone przez OWASP. Umożliwia przeprowadzanie testów dynamicznych, takich jak wykrywanie podatności typu: XSS, wstrzykiwanie kodu SQL czy niezabezpieczone bezpośrednie odwołania do obietku. Posiada przyjazny interfejs i wsparcie dla automatyzacji poprzez API.

- **Smart Scanner**: Narzędzie znane ze swojej prostoty i intuicyjnego interfejsu, ukierunkowane na podstawowe testy bezpieczeństwa. Chociaż mniej zaawansowane, zapewnia szybkie skanowanie aplikacji i wykrywanie typowych podatności.

Narzędzia zostały wybrane biorąc pod uwagę ich komplementarności – każde narzędzie ma specjalne właściwości, co pozwala na kompleksową analizę aplikacji w różnych aspektach. W dalszej części pracy przeanalizowane zostaną mocne i słabe strony tych narzędzi oraz ich efektywność w testach na przykładach własnej stworzonej aplikacji oraz WebGoat[13].

### *2.4 Kryteria oceny skuteczności narzędzi*

Narzędzia do skanowania podatności w aplikacjach webowych są bardzo ważnym element zabezpieczania oprogramowania. Aby efektywnie zabezpieczyć aplikacje, należy ocenić narzędzia pod kątem kilku istotnych kryteriów, takich jak zakres wykrywanych podatności, precyzja wyników, łatwość użytkowania, szybkość działania oraz jakość generowanych raportów. Przy wyborze narzędzi należy również brać pod uwagę ich zdolność do adaptacji i aktualizacji w obliczu dynamicznie zmieniających się zagrożeń oraz nowych technologii stosowanych w aplikacjach webowych. W tym badaniu narzędzia będą oceniane na podstawie poniższych kryteriów:

- Zakres wykrywanych podatności - jest to jeden z najważniejszych kryteriów oceny. Wskaźnik ten obejmuje zdolność narzędzia do wykrywania różnych typów podatności. Narzędzia powinny obejmować szeroki zakres zagrożeń. Narzędzia powinny skutecznie identyfikować najważniejsze podatności zgodnie z OWASP Top Ten, takie jak wstrzykiwanie kodu SQL, XSS, czy CSRF. Ważne jest również, aby narzędzia były w stanie wykryć podatności charakterystyczne dla danej technologii, np. podatności związane z bazami danych, frameworkami (np. Spring Boot, Angular), czy protokołami (HTTP, HTTPS).
- Precyzja wykrywania - wysoka dokładność wykrywania podatności jest kluczowym atrybutem skanerów. Tutaj można wyróżnić dwie grupy: Fałszywe pozytywy i Fałszywe negatywy. Pierwsza z nich to informacja, kiedy narzędzie zgłasza podatność, która w rzeczywistości nie występuje. Zbyt duża liczba fałszywych alarmów może prowadzić do utraty zaufania do wyników i utrudniać proces zabezpieczania aplikacji. Druga to przypadki, gdy narzędzie nie wykrywa istniejących luk bezpieczeństwa. Jest to bardziej niebezpieczne,

ponieważ może pozostawić aplikację podatną na ataki, mimo przekonania o jej bezpieczeństwie.

- Łatwość użycia - jest to cecha, która mówi, jak łatwo można wdrożyć i używać narzędzia. Kryteria obejmuje interfejs użytkownika, automatyzację i konfigurację z personalizacją. Przyjazny interfejs pozwala na bardziej intuicyjne uruchamianie skanów oraz lepszą analizę wyników. Ważnym elementem jest również możliwość integracji narzędzi z procesami CI/CD. Pozwala to na automatyczne skanowanie podczas procesu wdrażania aplikacji. Narzędzia powinny zatem być elastyczne i pozwalać na dostosowanie skanowania do specyficznych wymagań aplikacji.
- Wydajność i szybkość - wydajność narzędzi skanujących jest bardzo ważną cechą. Ma ona znaczenie zwłaszcza w dużych aplikacjach internetowych. Szybszy czas skanowania jest ważnym elementem szczególnie w środowiskach produkcyjnych. Narzędzie musi również być w stanie wykrywać podatności w dużych i złożonych projektach, przy zachowaniu wysokiej efektywności.
- Raportowanie i analiza wyników - jakość raportów generowanych przez narzędzie jest również bardzo ważnym elementem kryterium oceny. Raporty muszą być przejrzyste i zrozumiałe zarówno dla programistów, jak i zespołów bezpieczeństwa. Narzędzie musi dawać jasne raporty na temat wad i luk, jak również wskazywać, gdzie te luki się znajdują. Narzędzie musi klasyfikować podatności w zależności od poziomu ryzyka. Pomaga to programistom w określaniu priorytetu naprawy podatności.
- Możliwość aktualizacji - narzędzia do skanowania muszą być aktualizowane na bieżąco, ponieważ wraz z rozwojem technologii, podatności także ulegają zmianie. Narzędzie powinno regularnie aktualizować swoje bazy podatności oraz powinno nadążać za rozwojem nowych frameworków i technologii webowych.

# Część Badawcza

### *2.5 Przykłady scenariuszy testowych i kryteriów sukcesu*

**Scenariusz 1**: Test podatności wstrzykiwania kodu SQL

**Opis**: Przeprowadzenie testu z wykorzystaniem zapytań SQL w polach wejściowych (formularzach). Test sprawdza, czy dane użytkownika są nieodpowiednio filtrowane.

**Kroki testowe**:

1. Zaloguj się przy użyciu następującego łańcucha w polu z nazwą użytkownika i hasłem: ' OR '1'='1.
2. Sprawdź, czy dostęp do aplikacji został uzyskany bez podania poprawnych danych.

**Kryterium sukcesu**: Aplikacja nie powinna zaakceptować źle sformułowanego zapytania SQL.

**Scenariusz 2:** Test podatności wstrzykiwania skryptu do stron internetowych (XSS)

**Opis**: Sprawdzenie. czy aplikacja poprawnie filtruje dane wejściowe i zapobiega wstrzykiwaniu złośliwego kodu JavaScript.

**Kroki testowe**:

1. W formularzu dodawania użytkowników wprowadź tekst <script>alert('XSS')</script>.
2. Otwórz stronę z listą użytkowników i sprawdź, czy komunikat został wyświetlony.

**Kryterium sukcesu**: Kod JavaScript nie powinien zostać wykonany; dane powinny być zeskapowane.

**Scenariusz 3:** Test podatności niezabezpieczonego bezpośrednio odwołania do obiektu (IDOR)

**Opis**: Weryfikacja, czy użytkownik może uzyskać dostęp do zasobów, które są dla niego niewidoczne.

**Kroki testowe**:

1. Zaloguj się jako zwykły użytkownik.

2. Zmień adres URL w przeglądarce na /admin/users.
3. Sprawdź, czy uzyskujesz dostęp do panelu administratora.

**Kryterium sukcesu**: Zwykły użytkownik nie powinien mieć dostępu do zasobów administracyjnych.

**Scenariusz 4**: Brak wymuszania HTTPS

**Opis**: Test sprawdza, czy aplikacja wymusza korzystanie z szyfrowanego kanału do komunikacji HTTPS.

**Kroki testowe**:

1. Otwórz stronę aplikacji za pomocą HTTP.
2. Sprawdź, czy użytkownik jest automatycznie przekierowywany na HTTPS.

**Kryterium sukcesu**: Aplikacja powinna przekierować na HTTPS i nie dopuszczać transmisji przez HTTP.

**Scenariusz 5**: Brak odpowiednich nagłówków HTTP

**Opis**: Testowanie, czy aplikacja wysyła nagłówki ochrony, takie jak Content-Security-Policy, X-XSS-Protection, X-Content-Type-Options.

**Kroki testowe**:

1. Przeanalizuj odpowiedzi HTTP aplikacji.
2. Zidentyfikuj brakujące nagłówki.

**Kryterium sukcesu**: Wszystkie zalecane nagłówki powinny być obecne w odpowiedziach serwera.

**Scenariusz 6:** Brak ochrony przed zmuszaniem przeglądarki do wykonania złośliwej akcji (CSRF)

**Opis:** Weryfikacja, czy aplikacja chroni użytkowników przed atakami CSRF.

**Kroki testowe:**

1. Utwórz formularz HTML na zewnętrznej stronie, który wysyła żądanie POST do aplikacji.
2. Otwórz formularz w przeglądarce i sprawdź, czy żądanie zostanie zaakceptowane przez serwer.

**Kryterium sukcesu:** Serwer powinien odrzucić żądania z niewłaściwymi tokenami CSRF.

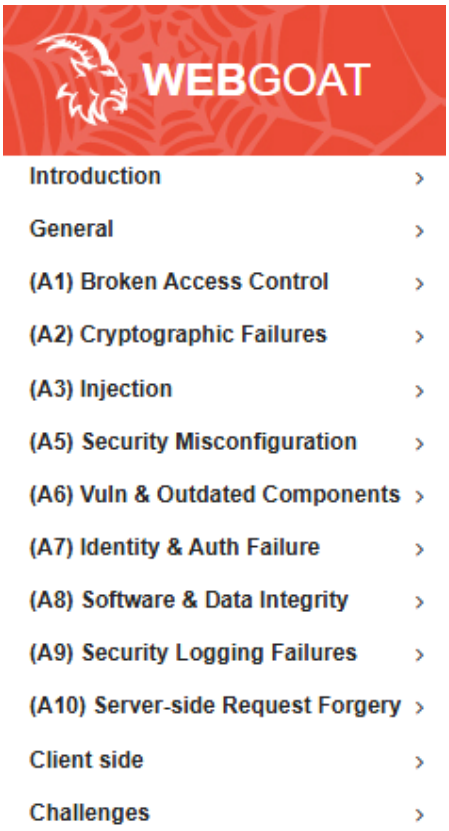
# 3. Testowane aplikacje

## *3.1 Opis i przedstawienie aplikacji*

W rozdziale tym zostaną opisane dwie aplikacje testowe wykorzystane do analizy skuteczności narzędzi do skanowania podatności. Obie aplikacje zostały celowo zaprojektowane lub skonfigurowane w sposób umożliwiający symulację rzeczywistych podatności, zgodnie z klasyfikacją OWASP Top Ten. Pierwsza z nich to **WebGoat**, narzędzie edukacyjne rozwijane przez OWASP. Drugą aplikacją jest prosta aplikacja do zarządzania projektami i zadaniami stworzona specjalnie do testów narzędzi zawierająca podstawowe luki bezpieczeństwa takie jak wstrzykiwanie kodu SQL, wstrzykiwanie skryptu do stron internetowych, obchodzenie ścieżki itp.

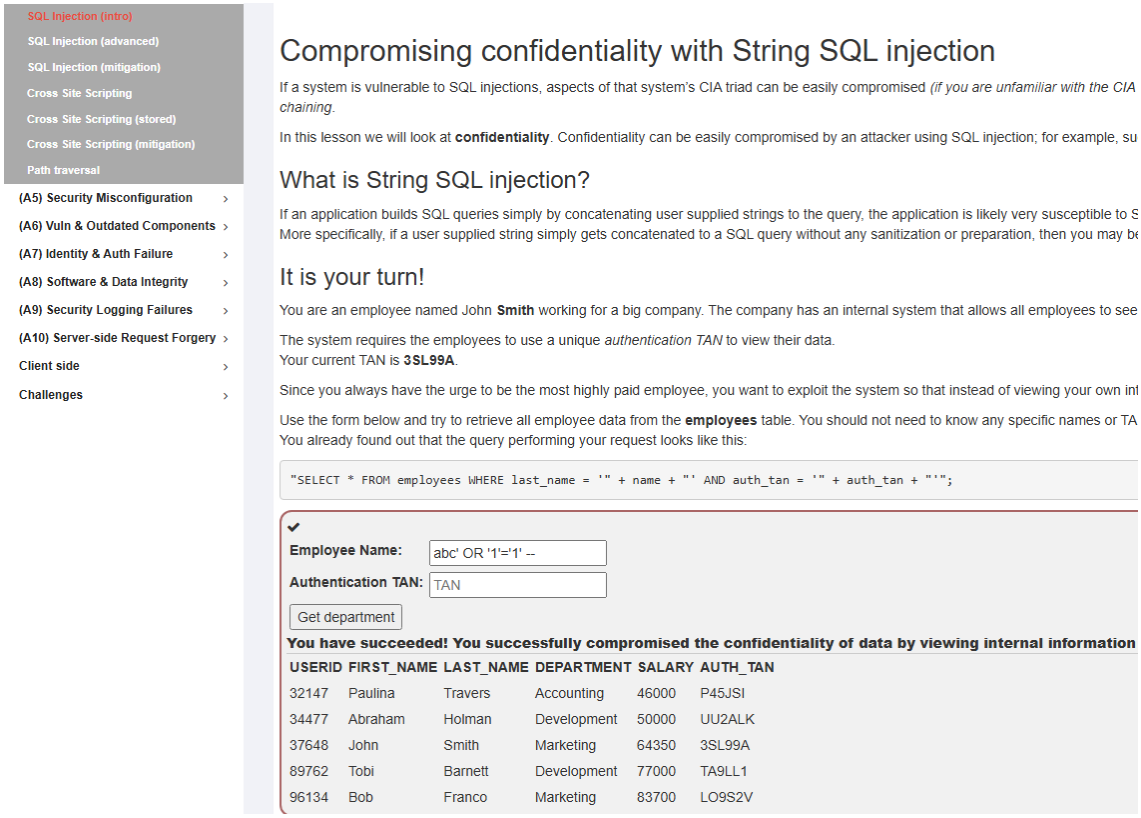
- WebGoat

Aplikacja WebGoat jest to aplikacja typu otwarto źródłowego stworzona do celów edukacyjnych oraz testowych posiadająca wszystkie podatności z listy OWSAP Top Ten. Idealnie sprawdza się jako punkt odniesienia w porównaniu skuteczności wykrywania podatności przez narzędzia skanujące.

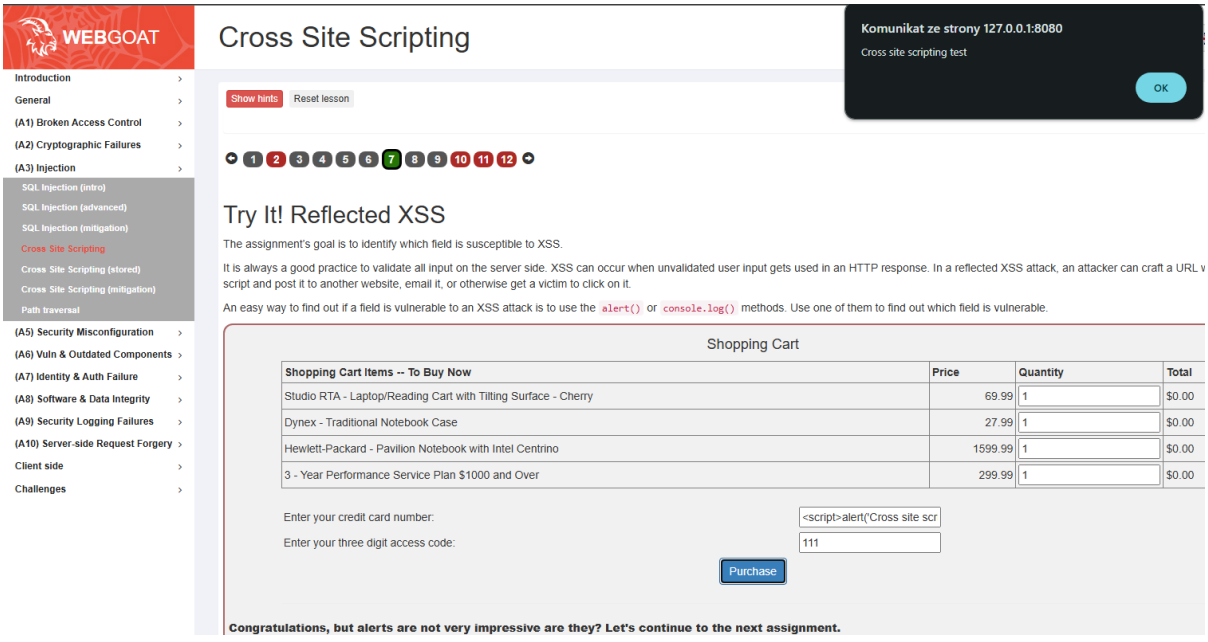



**Rys. 3.1** Menu aplikacji WebGoat[opracowanie własne]

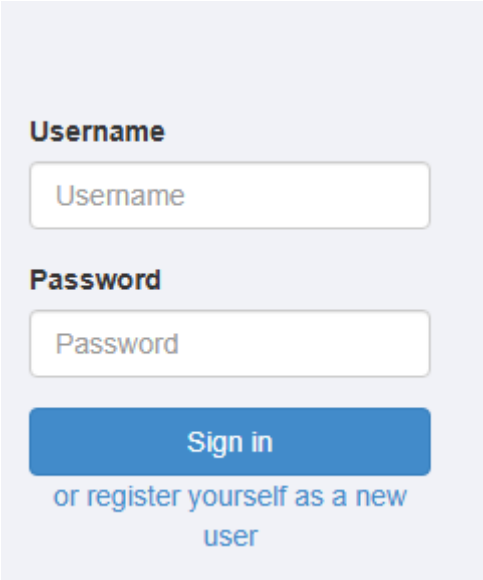
Posiada ona menu otwierające strony z poszczególnymi podatnościami do przetestowania (Rys. 3.1). Najbardziej interesujące dla nas pod względem testowania zautomatyzowanych narzędzi do wykrywania podatności jest CSRF i wstrzykiwanie (Rys. 3.2, Rys. 3.3) W zakładce tej znajdziemy strony z podatnościami takimi jak wstrzykiwanie kodu SQL oraz XSS. Aplikacja ta również posiada formularz logowania oraz rejestracji, które także są podatne na ataki (Rys. 3.4, Rys. 3.5).



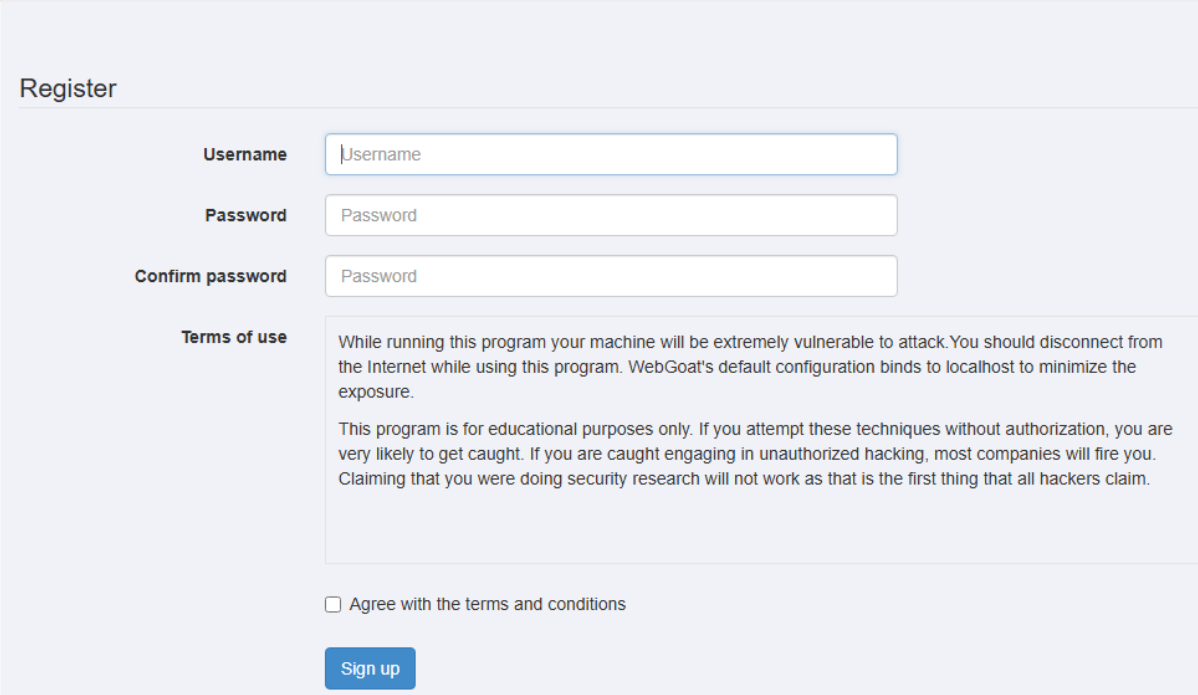

**Rys. 3.2** Przykładowy atak wstrzykiwania kodu SQL w aplikacji WebGoat[opracowanie własne]




**Rys. 3.3** Przykładowy atak XSS w aplikacji WebGoat[opracowanie własne]




**Rys. 3.4** Fromularz logowania aplikacji WebGoat[opracowanie własne]

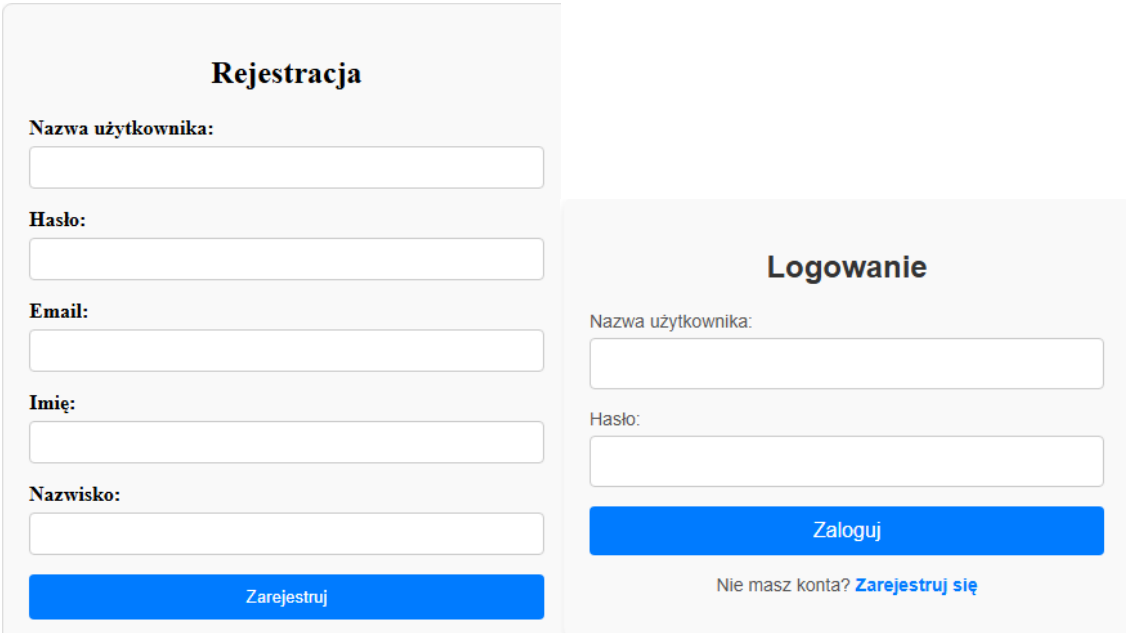



**Rys. 3.5** Formularz rejestracji do aplikacji WebGoat[opracowanie własne]

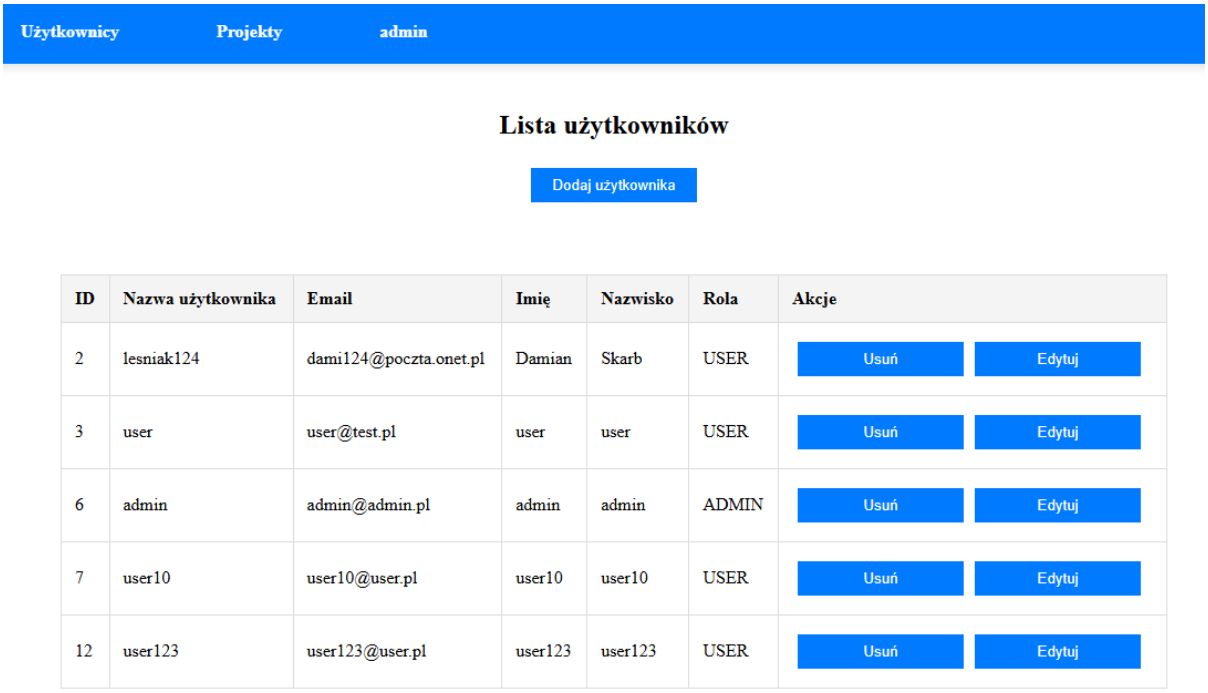
- Aplikacja do zarządzania projektami i zadaniami

Jest to prosta aplikacja webowa, która została zaprojektowana specjalnie z lukami bezpieczeństwa oraz podatnościami takimi jak wstrzykiwanie kodu SQL, XSS itp. Posiada

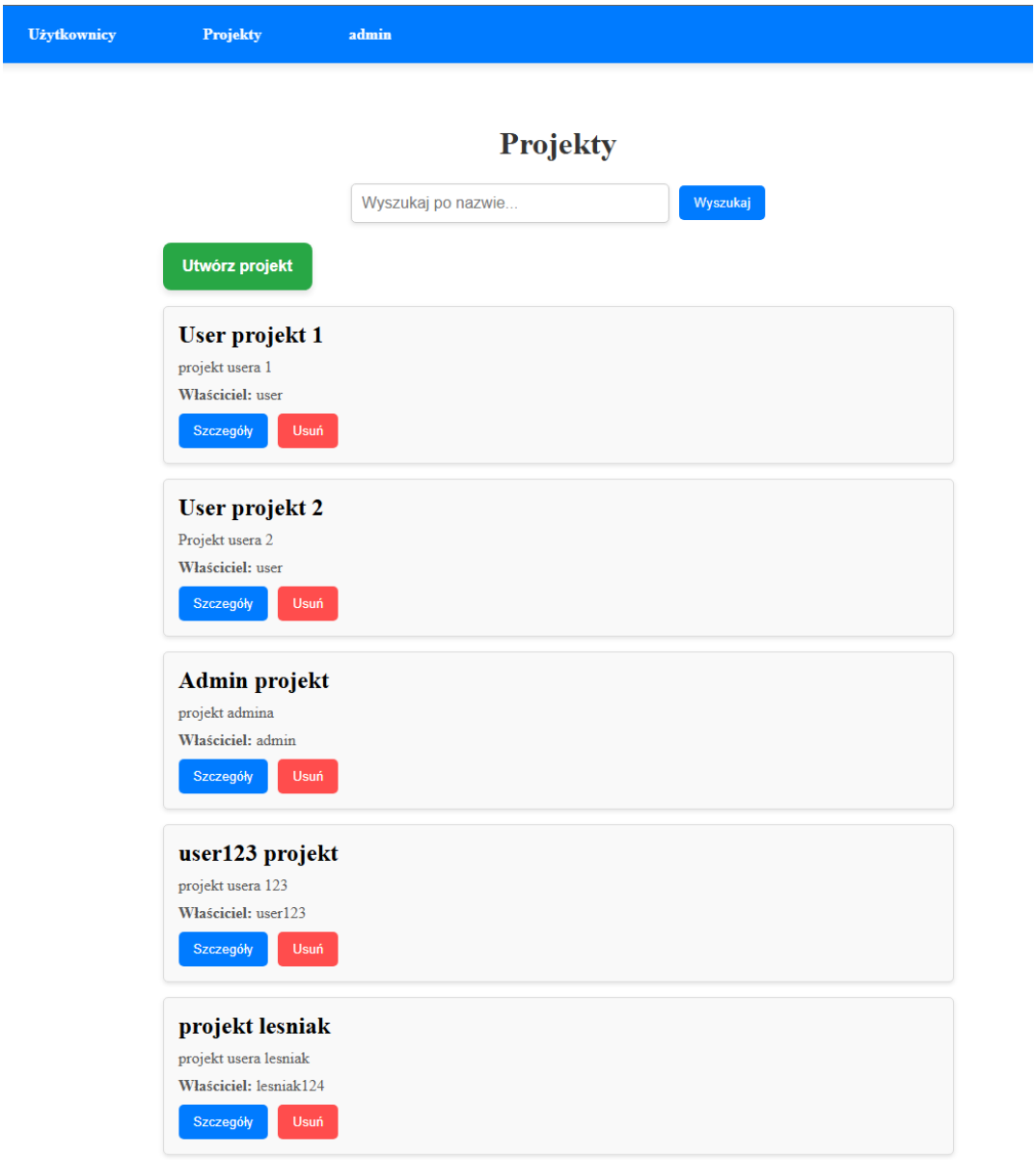
ona prosty formularz logowania i rejestracji (Rys. 3.6). Użytkownicy są podzieleni na role administratora i zwykłego użytkownika. Konto administratora wyróżnia się tym, że może dodawać nowych użytkowników (również administratorów), edytować ich oraz usuwać (Rys. 3.7, Rys. 3.9). Administrator widzi również projekty wszystkich użytkowników (Rys. 3.8), gdzie zwykły użytkownik widzi tylko swoje projekty i zadania. Administrator może również zarządzać projektami innych użytkowników. W aplikacji użytkownik może dodawać nowe projekty, usuwać stare oraz wyszukiwać swoje projekty po nazwie (Rys. 3.10, Rys. 3.11). W podglądzie projektu, użytkownik ma możliwość edycji nazwy oraz opisu projektu, a także może dodawać, usuwać lub oznaczać zadania jako wykonane lub niewykonane (Rys. 3.12). Wszystkie formularze oraz pola tekstowe są podatne na ataki typu SQL Injection oraz Cross site scripting. Nie ma również wprowadzonej kontroli dostępu do zasobów oraz ścieżek, dzięki czemu zwykły użytkownik może przejść do ścieżki, która powinna być widoczna tylko administratorowi. Architektura aplikacji składa się z trzech części: bazy danych, aplikacji serwerowej, aplikacji frontendowej. Wykorzystana baza danych to MySQL, aplikacja serwerowa została napisana w Javie z wykorzystaniem Spring Framework, natomiast aplikacja użytkownika została zaprojektowana w Angularze. Aplikacje komunikują się ze sobą za pomocą REST API. W aplikacji serwerowej wszystkie zabezpieczenia z wyjątkiem autoryzacji użytkowników zostały wyłączone na potrzeby testów.



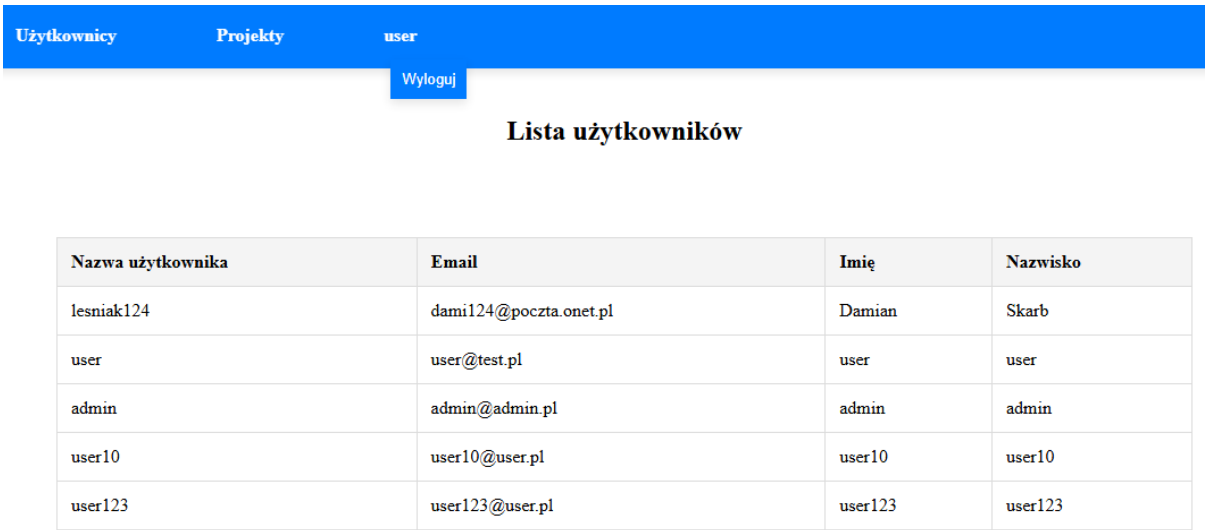

**Rys. 3.6** Fromularz logowania i rejestracji aplikacji do zarządzania projektami[opracowanie własne]



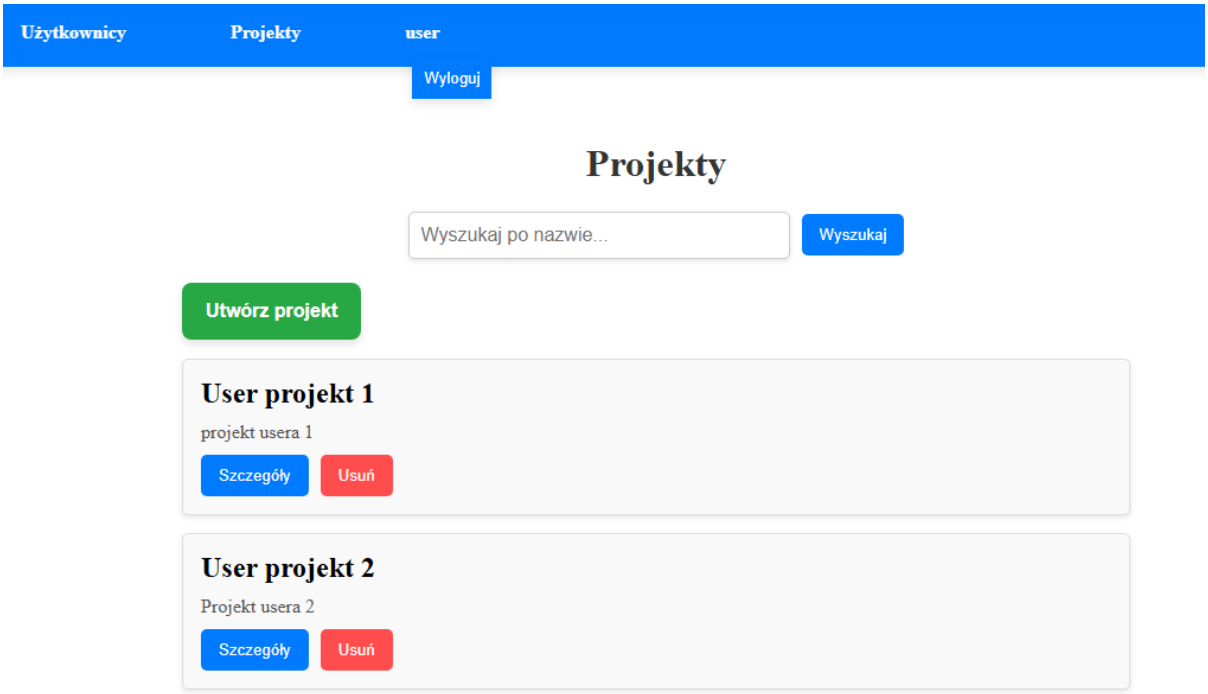

**Rys. 3.7** Widok listy użytkowników z konta admina[opracowanie własne]




**Rys. 3.8** Widok listy projektów z konta admina[opracowanie własne]

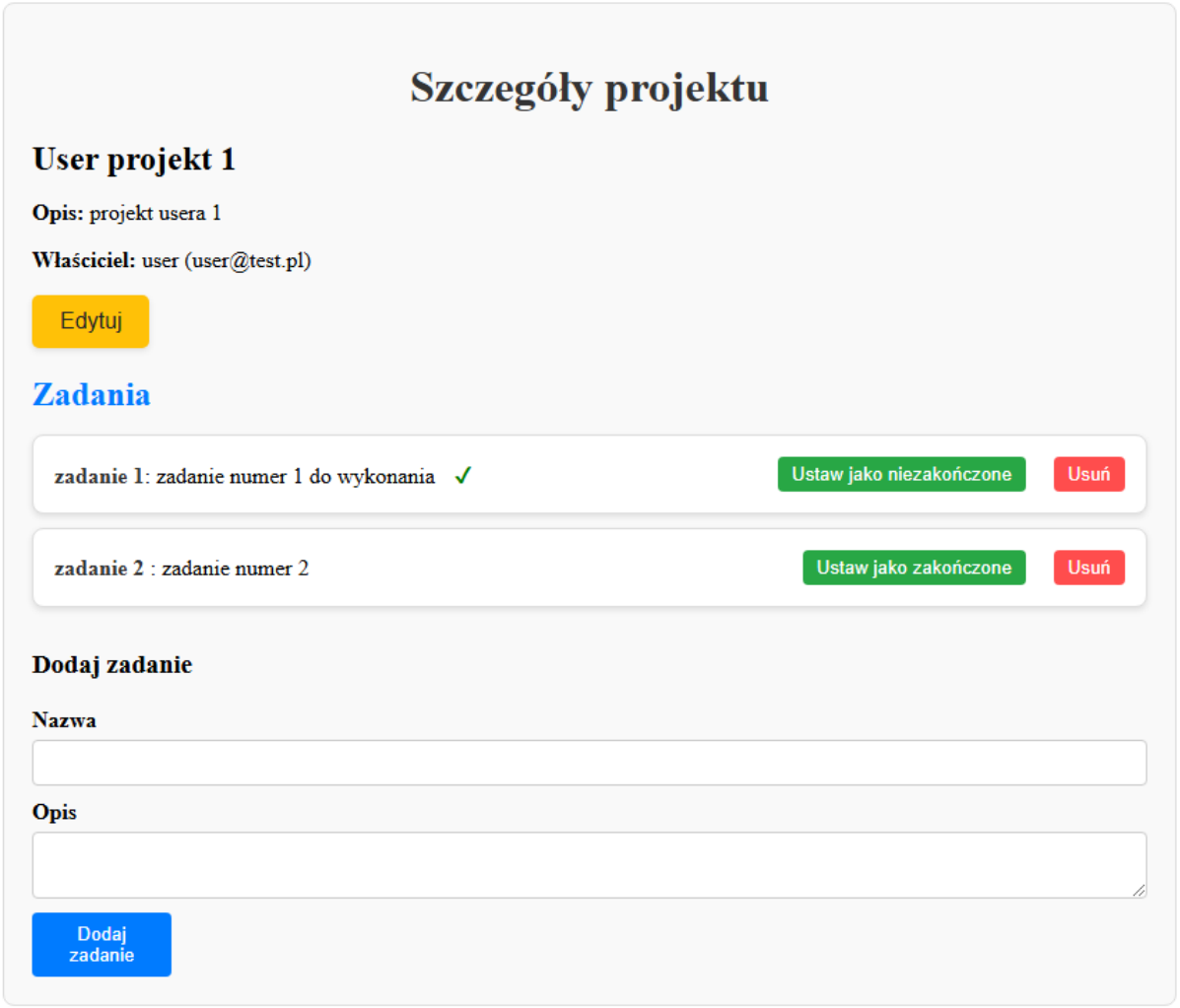



**Rys. 3.9** Widok listy użytkowników z konta zwykłego użytkownika[opracowanie własne]




**Rys. 3.10** Widok listy projektów z konta zwykłego użytkownika[opracowanie własne]


**Rys. 3.11** Formularz dodawania projektu[opracowanie własne]




**Rys. 3.12** Widok szczegółów projektu[opracowanie własne]

# 4. Analiza skuteczności narzędzi

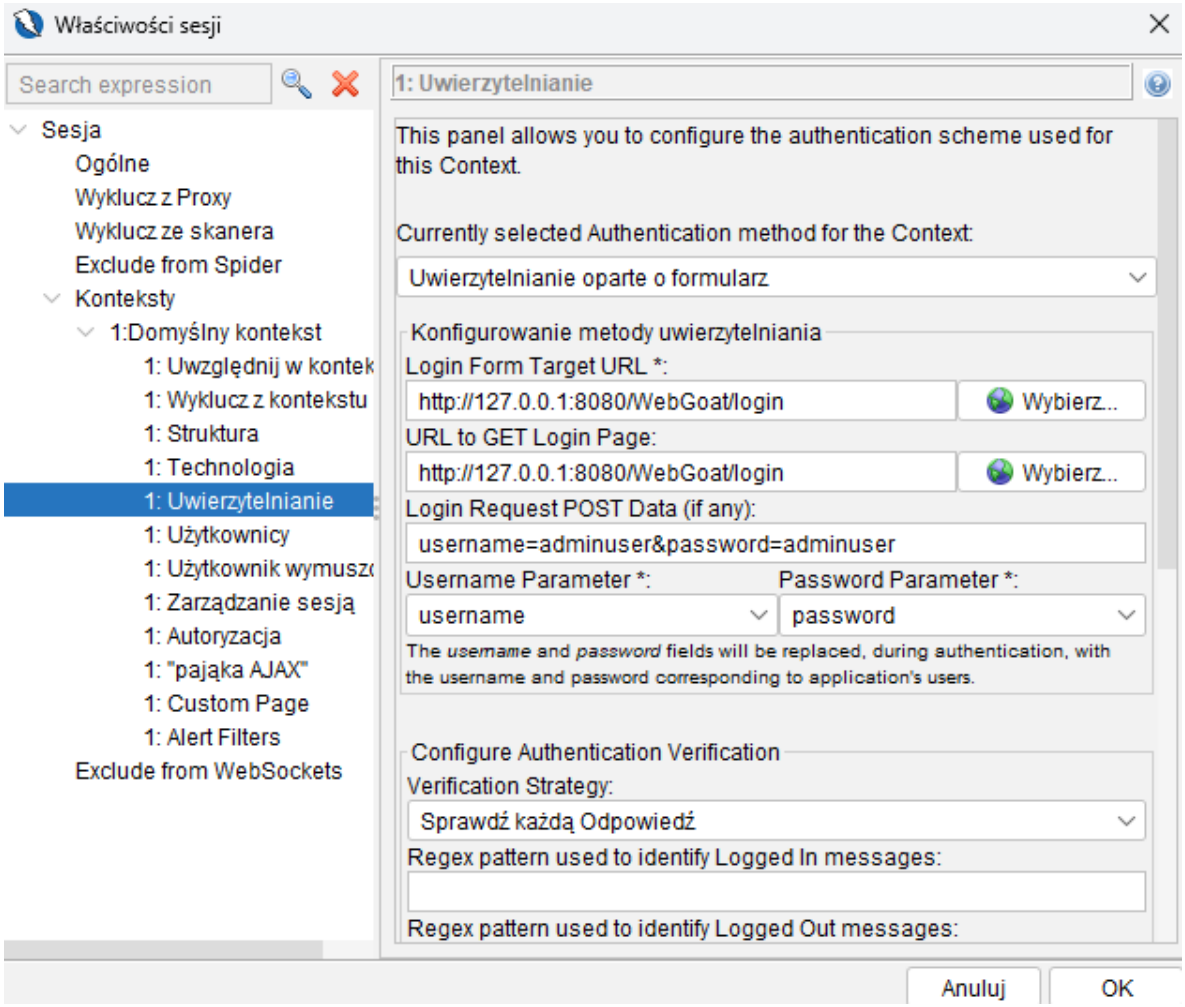
## *4.1 Przeprowadzenie testów na różnych aplikacjach webowych*

W ramach tego etapu pracy przeprowadzono szczegółowe testy bezpieczeństwa na wybranych aplikacjach webowych: WebGoat i aplikacja do zarządzania projektami. Analizowano skuteczność trzech narzędzi skanujących: **Burp Suite**, **OWASP ZAP** i **Smart Scanner**, w wykrywaniu podatności. Obie aplikację zostały uruchomione lokalnie na systemie Windows. Aplikacja WebGoat została uruchomiona na porcie 8080 natomiast druga aplikacja na porcie 4200.

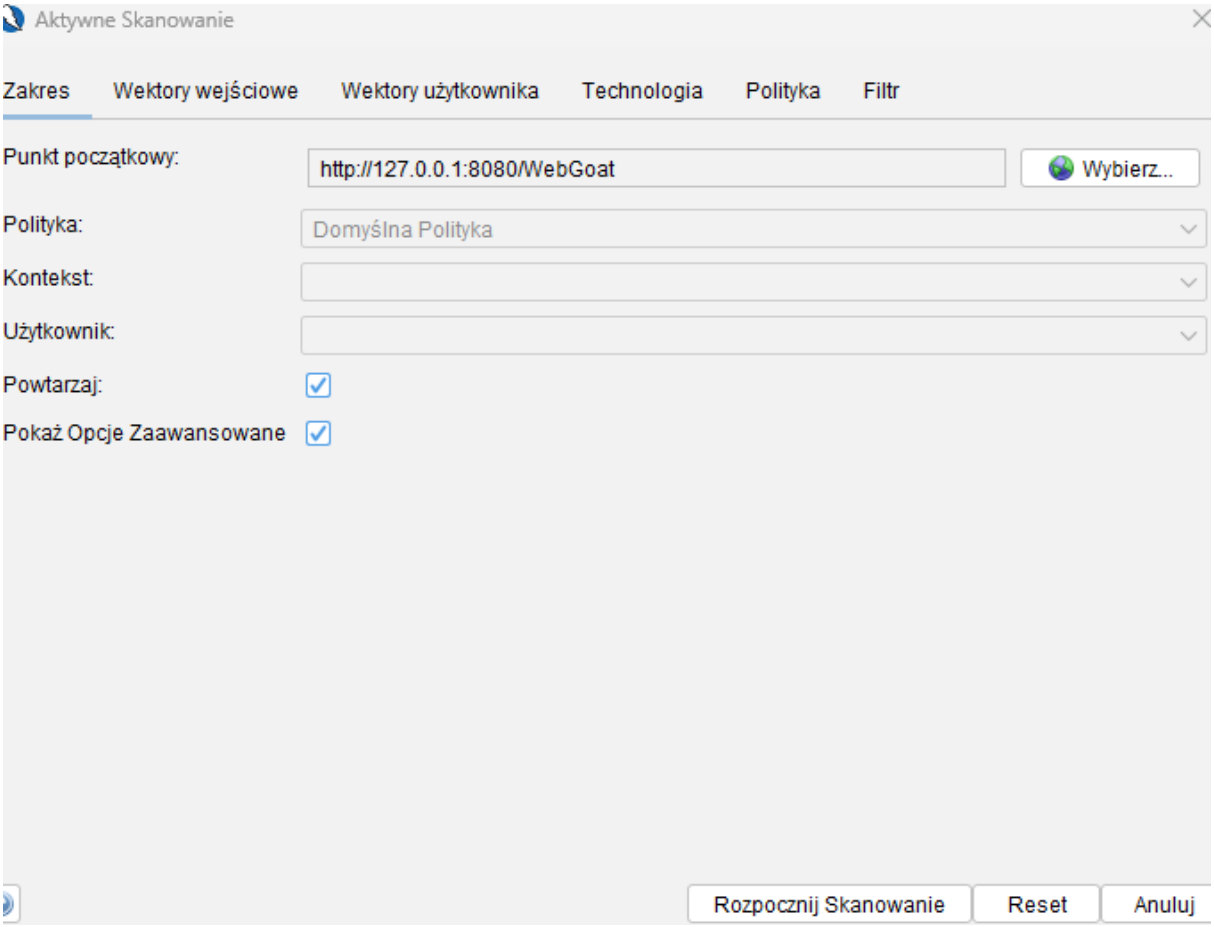
Konfiguracja narzędzi:

- OWSAP ZAP

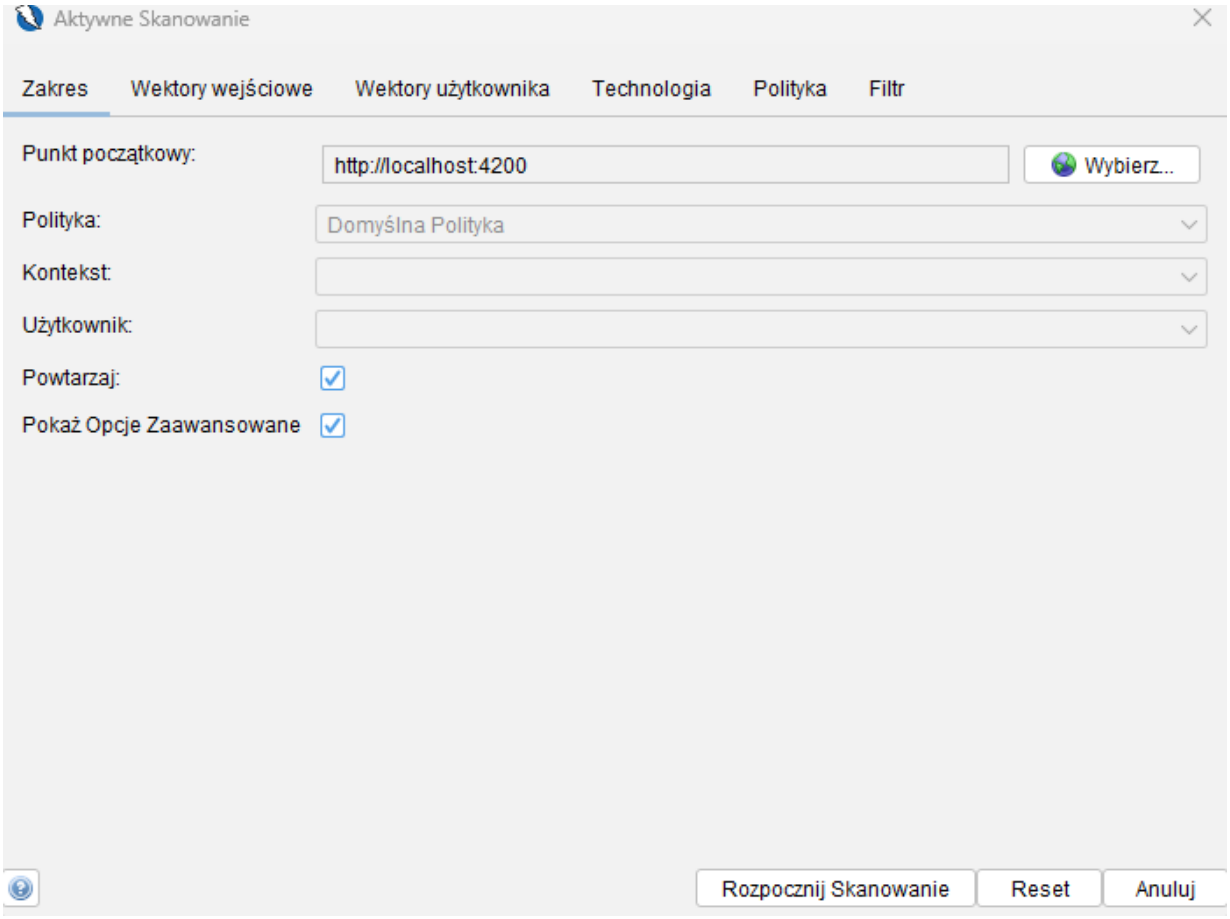
Na początku zostanie przeskanowana aplikacja WebGoat przy użyciu narzędzia OWSAP ZAP. Jak widać na rysunku (Rys 4.1), najpierw zostanie skonfigurowana nasza sesja. Ponieważ aplikacje są chronione przed nieautoryzowanymi użytkownikami, dlatego w konfiguracji zostało ustawione uwierzytelnianie jako formularz oraz został dostarczony użytkownik istniejący w systemie wraz z adresem URL do formularza logowania. Następnie została ustawiona polityka skanowania, czyli zostały włączone wszystkie możliwe podatności, które mogą być wykryte z odpowiednimi poziomami ryzyka oraz został podany punkt startowy jako URL aplikacji tak jak na rysunkach (Rys 4.2, Rys. 4.3), od którego ma zostać rozpoczęte skanowanie, tak aby cała aplikacja została sprawdzona. Po rozpoczęciu skanowania, narzędzie najpierw szuka wszystkich dostępnych ścieżek oraz formularzy w aplikacji tworząc mapę, a następnie dokonuje sprawdzenia każdego znalezionego URL'a pod względem występowania podatności zdefiniowanych w polityce skanowania.




**Rys. 4.1** narzędzia OWSAP Konfiguracja ZAP 1[opracowanie własne]



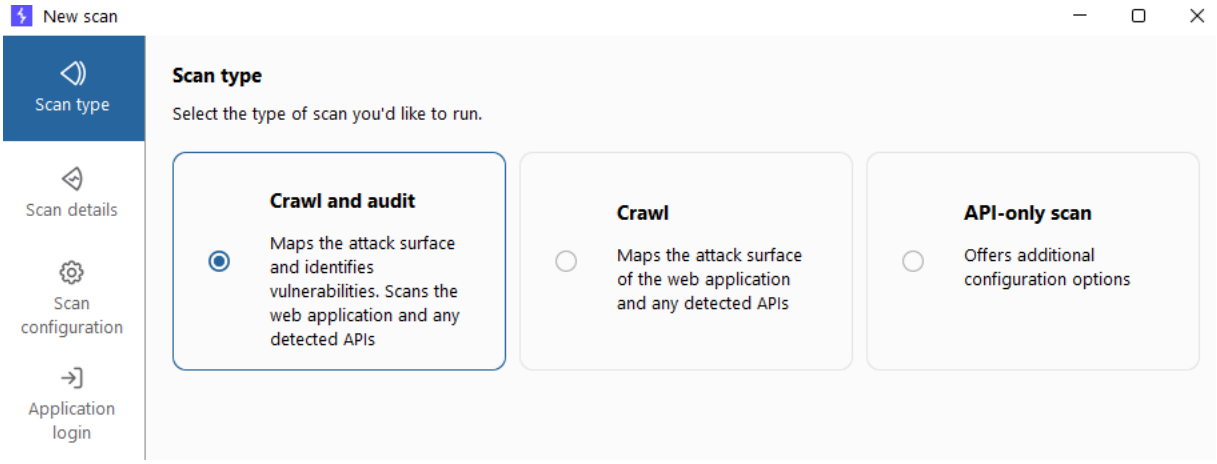

**Rys. 4.2** Konfiguracja narzędzia OWSAP ZAP 2[opracowanie własne]



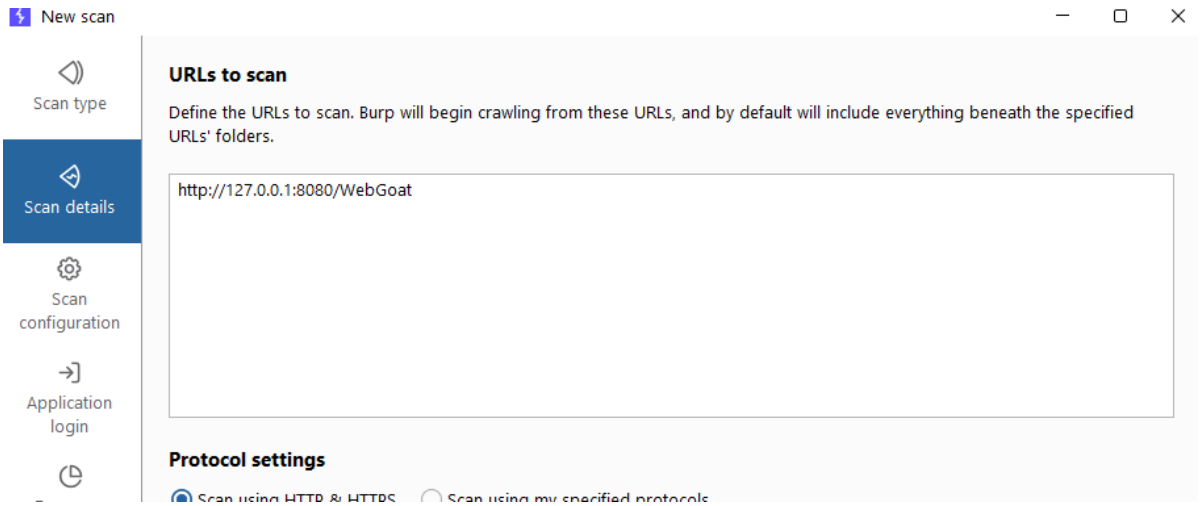

**Rys. 4.3** Konfiguracja narzędzia OWSAP ZAP 3[opracowanie własne]

- Burp Suite

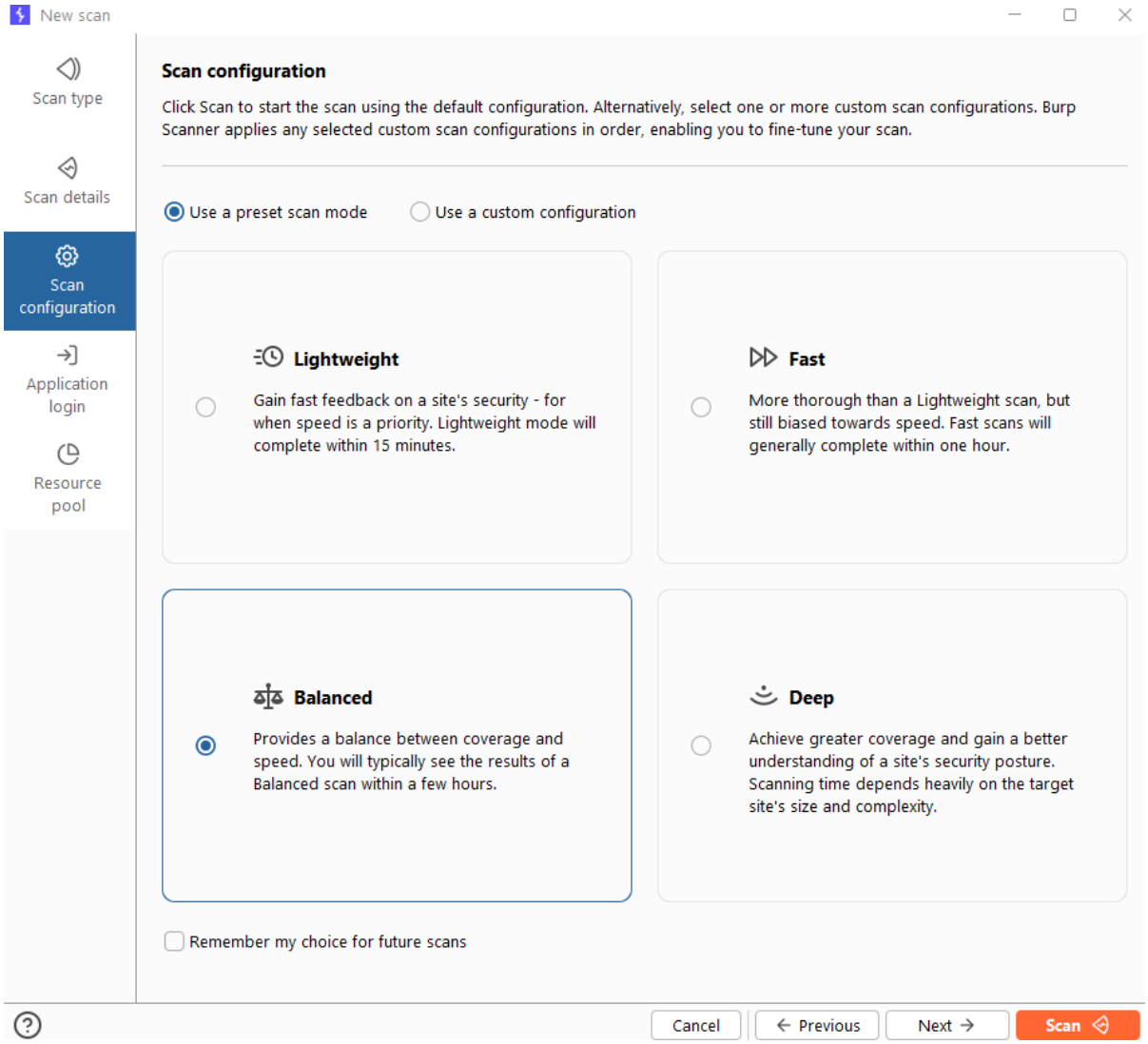
Następnie zostało skonfigurowane narzędzie Burp Suite. W tym wypadku na początku został wybrany rodzaj skanowania Crawl and audit, czyli najpierw aplikacja zostanie przeszukana w celu znalezienia wszystkich dostępnych URL'i a następnie każdy punkt końcowy aplikacji zostanie sprawdzony w celu znalezienia podatności (Rys 4.4). Następnie został ustawiony punkt startowy skanowania tak jak na rysunku (Rys 4.5). W kolejnym kroku zostało skonfigurowane samo skanowanie. Narzędzie Burp Suite ma do wyboru 4 różne rodzaje skanowania, które różnią się od siebie dokładnością oraz szybkością. W celach testowych zostało skanowanie zbalansowane, które jest kompromisem pomiędzy dokładnością a szybkością (Rys 4.6), ponieważ głębokie (najbardziej dokładne) skanowanie może trwać nawet kilka godzin. Na samym końcu został dodany użytkownik do autoryzacji w aplikacji (Rys 4.7).



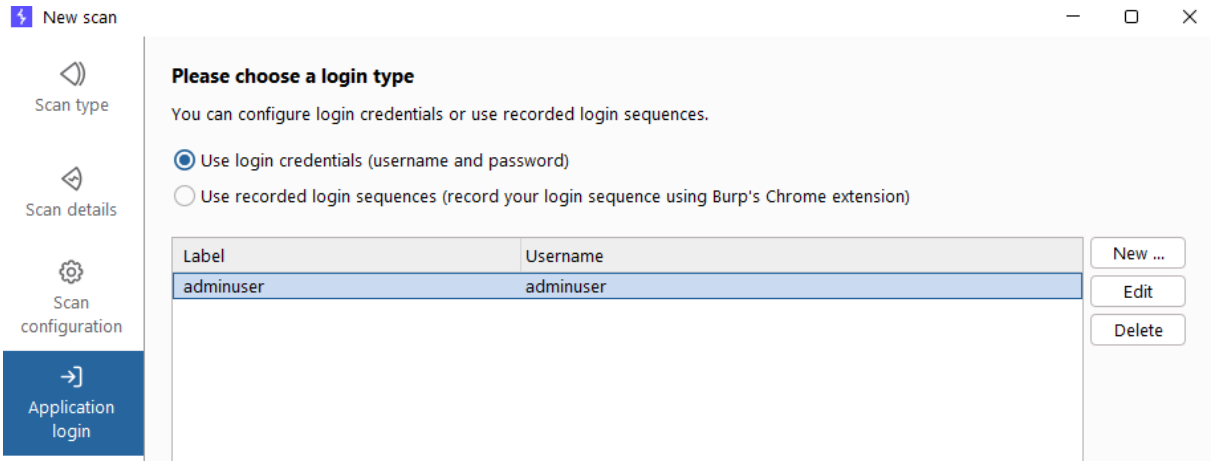

**Rys. 4.4** Konfiguracja narzędzia Burp Suite 1[opracowanie własne]




**Rys. 4.5** Konfiguracja narzędzia Burp Suite 2[opracowanie własne]



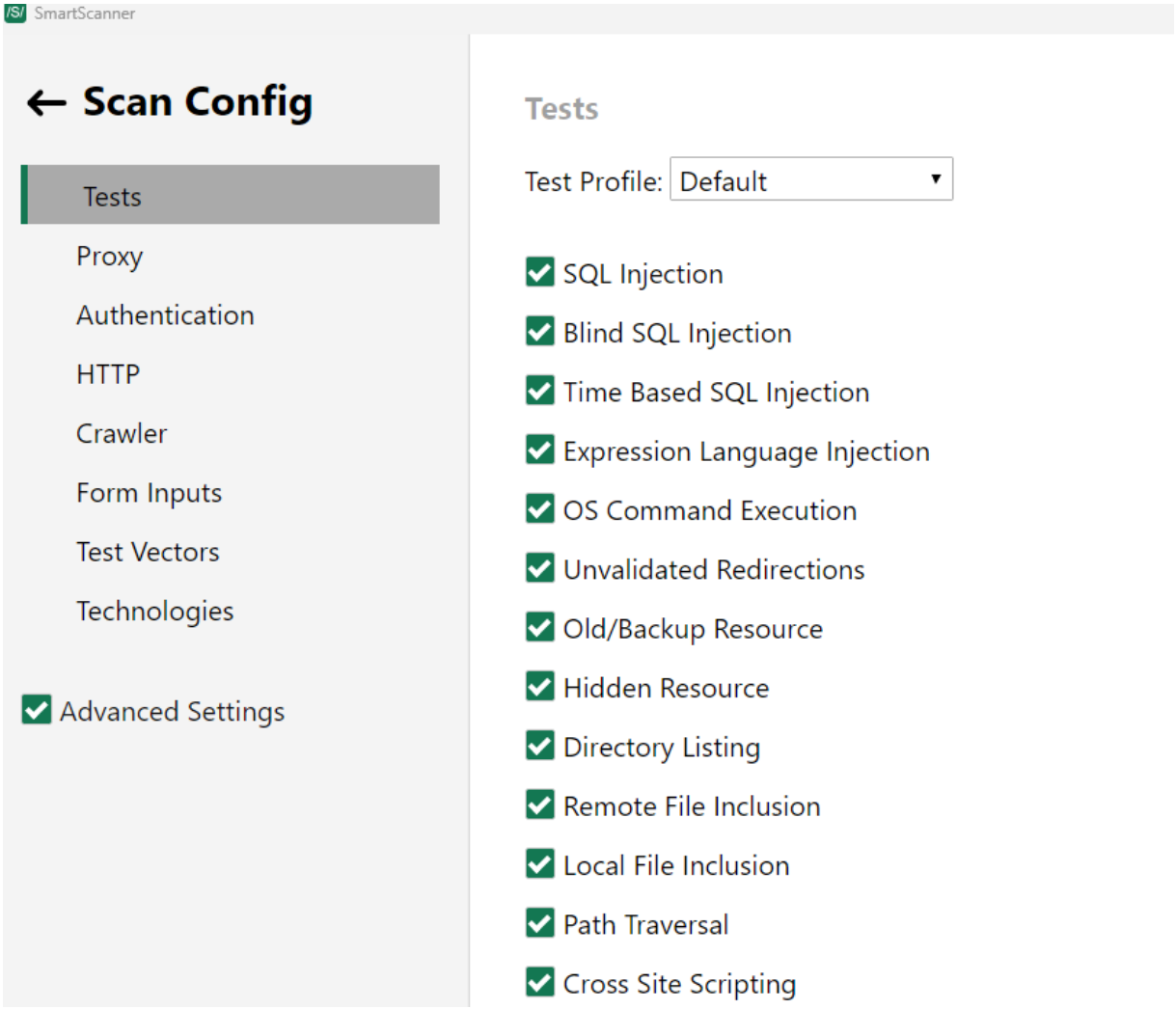

**Rys. 4.6** Konfiguracja narzędzia Burp Suite 3[opracowanie własne]



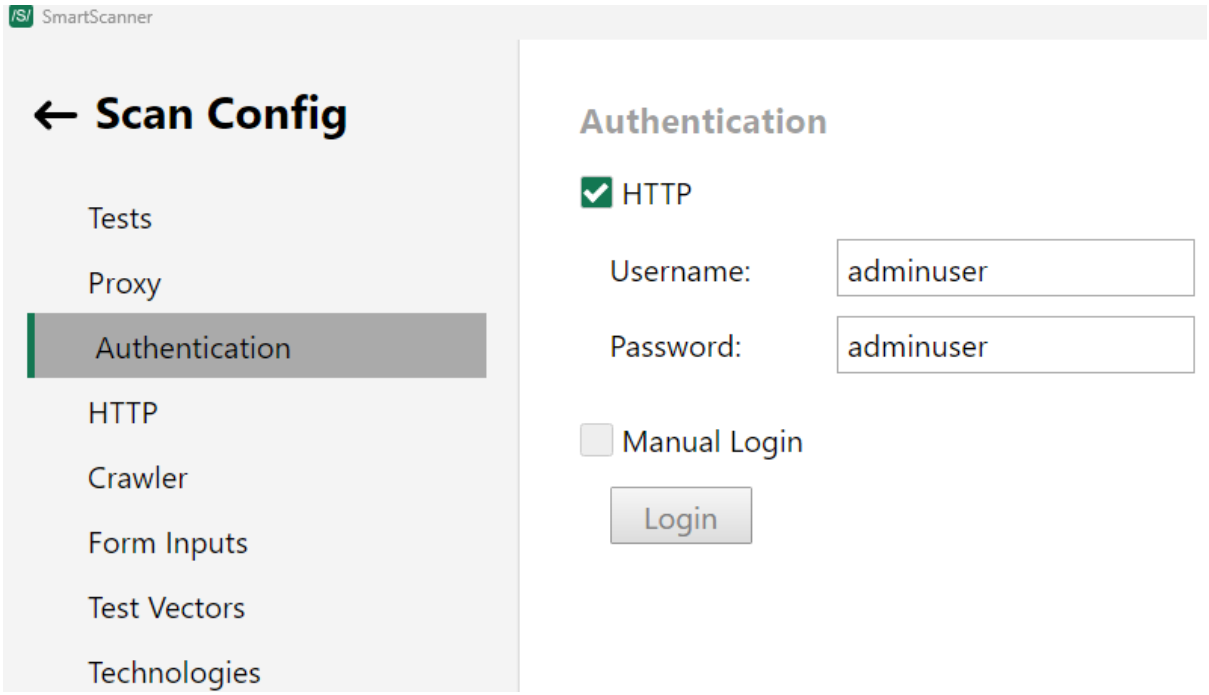

**Rys. 4.7** Konfiguracja narzędzie Burp Suite 4[opracowanie własne]

- Smart Scanner

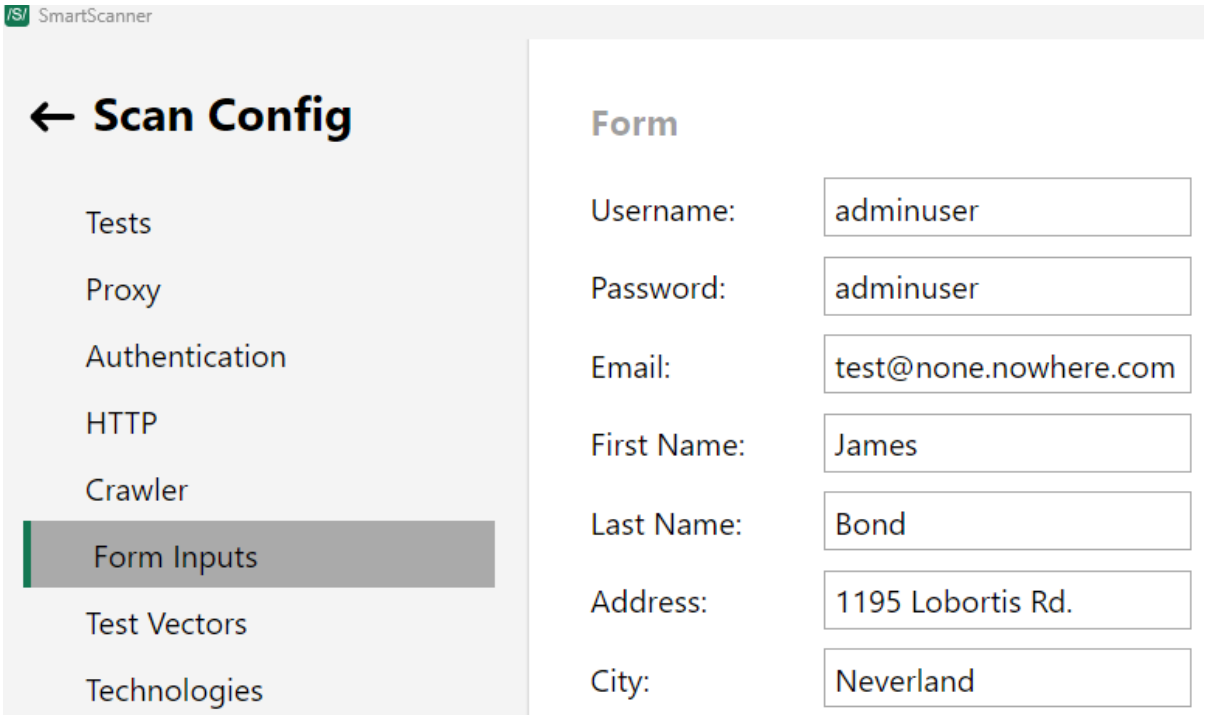
Ostatnim skonfigurowanym narzędziem jest Smart Scanner, który posiada najbardziej ubogą konfigurację. Na początku zostały wybrane rodzaje podatności, które zostaną szukane (Rys 4.8). Kolejnym krokiem było ustawienie użytkownika do autoryzacji (Rys 4.9), a w ostatnim kroku zostały ustawione dane, które będą wpisywane do napotkanych formularzy w aplikacji (Rys 4.10). Jest to unikalna opcja w 3 wybranych narzędziach.




**Rys. 4.8** Konfiguracja narzędzia Smart Scanner 1[opracowanie własne]




**Rys. 4.9** Konfiguracja narzędzia Smart Scanner 2[opracowanie własne]




**Rys. 4.10** Konfiguracja narzędzia Smart Scanner 3[opracowanie własne]

## *4.2 Zestawienie wyników skuteczności każdego narzędzia*

W tym podrozdziale przeprowadzono zestawienie wyników skuteczności narzędzi do skanowania podatności na podstawie testów wykonanych na aplikacjach testowych. Analizie poddano dane zebrane za pomocą narzędzi Burp Suite, OWSAP ZAP oraz Smart Scanner.

Uwzględnia ona liczbę wykrytych podatności, ich poziom ryzyka oraz czas skanowania. Celem tego porównania jest określenie mocnych i słabych stron każdego narzędzia oraz ocena ich przydatności w zależności od aplikacji.

Raporty każdego narzędzia z przeprowadzonych testów na obu aplikacjach przedstawione są na rysunkach (Rys. 4.11, Rys. 4.12, Rys. 4.13, Rys. 4.14, Rys.4.15, Rys. 4.16).

**Alert counts by alert type**

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

| Alert type | Risk | Count |
|---|---|---|
| External Redirect | Wysoki | 1 (4,3%) |
| Obchodzenie Ścieżki | Wysoki | 13 (56,5%) |
| SQL Injection | Wysoki | 8 (34,8%) |
| SQL Injection - Hypersonic SQL | Wysoki | 3 (13,0%) |
| Absence of Anti-CSRF Tokens | Średni | 1 (4,3%) |
| Content Security Policy (CSP) Header Not Set | Średni | 15 (65,2%) |
| Format String Error | Średni | 5 (21,7%) |
| Missing Anti-clickjacking Header | Średni | 16 (69,6%) |
| Spring Actuator Information Leak | Średni | 1 (4,3%) |
| Vulnerable JS Library | Średni | 1 (4,3%) |
| Cookie No HttpOnly Flag | Niski | 2 (8,7%) |
| Cookie without SameSite Attribute | Niski | 2 (8,7%) |
| Cross Site Scripting Weakness (Persistent in JSON Response) | Niski | 1 (4,3%) |

**Rys. 4.11** Lista podatności w raporcie OWSAP ZAP aplikacji WebGoat[opracowanie własne]

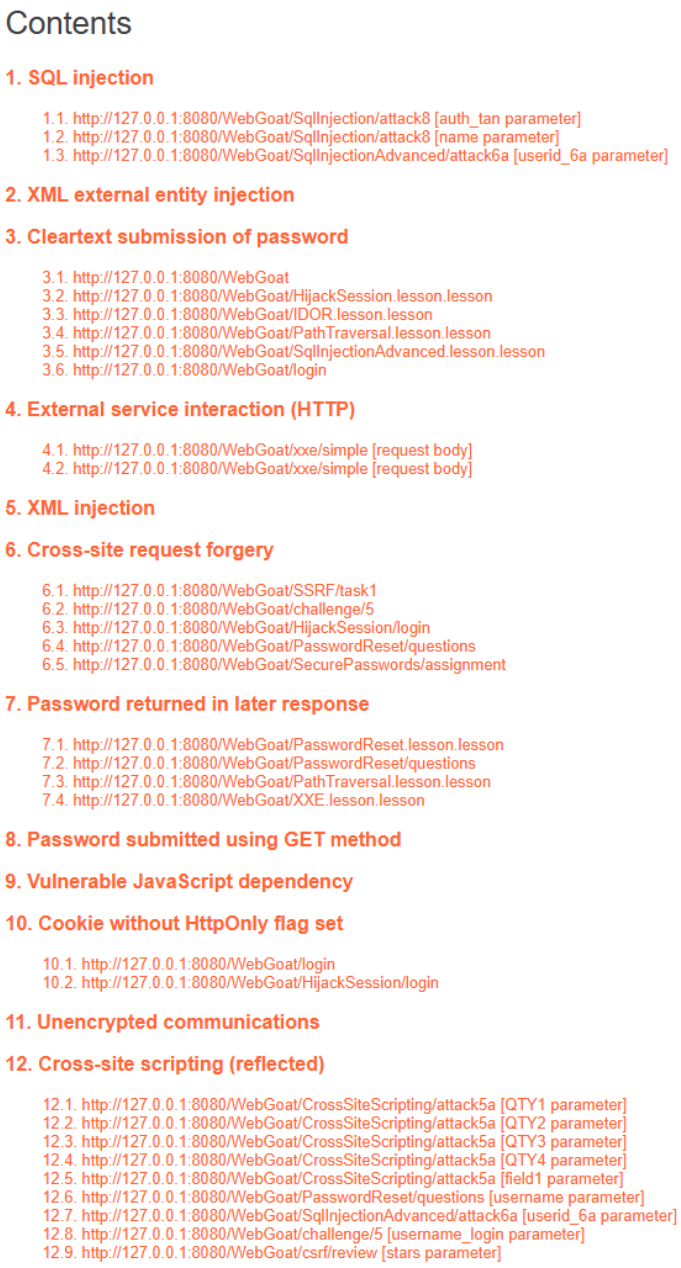
## Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

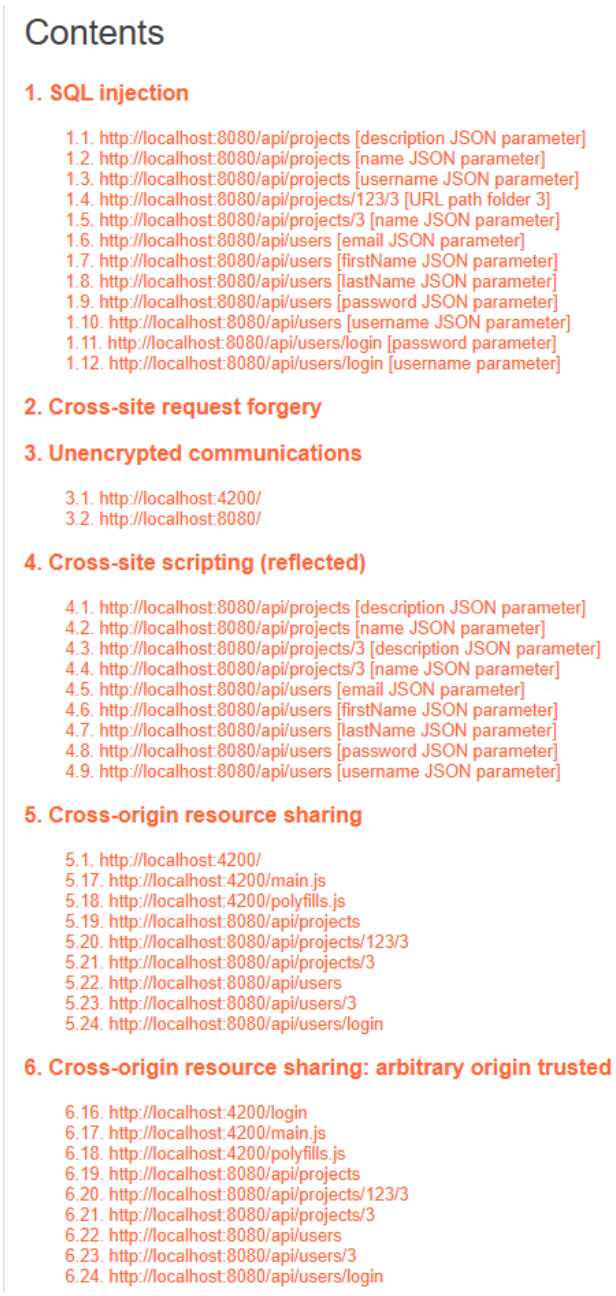
(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

| Alert type | Risk | Count |
|---|---|---|
| Cloud Metadata Potentially Exposed | Wysoki | 1 (7,7%) |
| Obchodzenie Ścieżki | Wysoki | 1 (7,7%) |
| SQL Injection | Wysoki | 4 (30,8%) |
| SQL Injection - MySQL | Wysoki | 4 (30,8%) |
| CSP: Wildcard Directive | Średni | 2 (15,4%) |
| Content Security Policy (CSP) Header Not Set | Średni | 5 (38,5%) |
| Cross-Domain Misconfiguration | Średni | 24 (184,6%) |
| Missing Anti-clickjacking Header | Średni | 5 (38,5%) |
| Przepełnienie bufora | Średni | 3 (23,1%) |
| X-Content-Type-Options Header Missing | Niski | 23 (176,9%) |

**Rys. 4.12** Lista podatności w raporcie OWSAP ZAP aplikacji do zarządzania projektami[opracowanie własne]




**Rys. 4.13** Lista podatności w raporcie Burp Suite aplikacji WebGoat[opracowanie własne]




**Rys. 4.14** Lista podatności w raporcie Burp Suite aplikacji do zarządzania projektami[opracowanie własne]

**List of Issues**

1– Password Sent Over HTTP
- 1.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10
- 1.2– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

2– Internal Server Error
- 2.1– http://127.0.0.1:8080/WebGoat/start.mvc?lang=smta < b"c'detms769#lesson/WebGoatIntroduction.lesson
- 2.2– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

3– No Redirection from HTTP to HTTPS
- 3.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

4– Unreferenced Login Page Found
- 4.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

5– Medium Impact Issue
- 5.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

6– No HTTPS
- 6.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

7– Auto Complete Enabled Password Input
- 7.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10
- 7.2– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

8– Content-Security-Policy Header is Missing
- 8.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

9– X-Frame-Options Header is Missing
- 9.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

10– Application Error
- 10.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

11– Unreferenced Resource Found
- 11.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10
- 11.2– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10
- 11.3– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

12– X-Content-Type-Options Header is Missing
- 12.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

13– Referrer-Policy Header is Missing
- 13.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

14– Email Address Disclosure
- 14.1– http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10

15– Target Information
- 15.1– http://127.0.0.1:8080

**Rys. 4.15** Lista podatności w raporcie Smart Scanner aplikacji WebGoat[opracowanie własne]

**List of Issues**

1– No Redirection from HTTP to HTTPS

1.1– http://localhost:4200/login

2– No HTTPS

2.1– http://localhost:4200/login

3– Content-Security-Policy Header is Missing

3.1– http://localhost:4200/login

4– X-Frame-Options Header is Missing

4.1– http://localhost:4200/login

5– X-Content-Type-Options Header is Missing

5.1– http://localhost:4200/login

6– Cross-Origin Resource Sharing Allowed

6.1– http://localhost:4200/login

7– Referrer-Policy Header is Missing

7.1– http://localhost:4200/login

8– Broken Link

8.1– http://localhost:4200/

**Rys. 4.16** Lista podatności w raporcie Smart Scanner aplikacji do zarządzania projektami[opracowanie własne]

**Tabela 4.1** Podatności wykryte przez poszczególne narzędzie w aplikacji WebGoat

| Znalezione podatnościw aplikacji WebGoat | OWSAP ZAP | Burp Suite | Smart Scanner |
|---|---|---|---|
| SQL Injection | TAK | TAK | NIE |
| Obchodzenie ścieżki | TAK | NIE | NIE |
| Brak nagłówka CSP | TAK | NIE | TAK |
| Zewnętrzne przekierowanie | TAK | TAK | NIE |
| Brak nagłówka Anti-clickjacking | TAK | TAK | TAK |
| XML Injection | NIE | TAK | NIE |
| Przekazywanie hasła jako jawny tekst | NIE | TAK | TAK |
| Cross-Site Scripring | TAK | TAK | NIE |
| Cross-Site request forgery | TAK | TAK | NIE |
| Brak zakodowanej komunikacji | NIE | TAK | TAK |

**Tabela 4.2** Podatności wykryte przez poszczególne narzędzie w aplikacji do zarządzania projektami

| Znalezione podatnośći w aplikacji do zarządzania projektami | OWSAP ZAP | Burp Suite | Smart Scanner |
|---|---|---|---|
| SQL Injection | TAK | TAK | NIE |
| Cloud meta data exposed | TAK | NIE | NIE |
| Obchodzenie ścieżki | TAK | NIE | NIE |
| Brak nagłówka CSP | TAK | NIE | TAK |
| Cross-Site scripring | TAK | TAK | NIE |
| Cross-Site request forgery | NIE | TAK | NIE |
| Brak zakodowanej komunikacji | NIE | TAK | TAK |
| Brak nagłówka Anti-clickjacking | TAK | NIE | TAK |

- Wyniki w aplikacji WebGoat

Wykryte podatności przez narzędzia są podzielone według poziomu ryzyka: wysoki, średni i niski poziom. OWSAP ZAP wykrył 16 alertów, w tym 4 o wysokim poziomie ryzyka, 6 o średnim i 6 o niskim. Najwięcej alertów wykrytych przez to narzędzie to wstrzykiwanie kodu SQL – w sumie 11 miejsc wystąpienia, oraz obchodzenie ścieżki – w sumie 13 miejsc wystąpienia. Czas skanowania wyniósł w przybliżeniu 10 minut. Wygenerowany raport jest czytelny i bardzo dokładny. Zawiera on dokładne miejsce występowania podatności wraz z wysłanym żądaniem, szczegółowym opisem luki oraz wskazówkami do rozwiązania problemu (Rys. 4.17, Rys. 4.18, Rys. 4.19). Raport posiada również tabelę podsumowującą wszystkie alerty z podziałem na ryzyko i pewność wystąpienia, co pozwala szybko oszacować stan aplikacji (Rys 4.20). Pełna lista wykrytych podatności przez OWSAP ZAP wraz z porównaniem do pozostałych narzędzi została przedstawiona w tabeli (Tabela 4.1) oraz na rysunku (Rys. 4.11). Jak widać poradził on sobie porównywalnie do Burp Suite w rodzaju wykrytych podatności, który wykrył również 4 alerty o wysokim ryzyku, jednak tylko 4 o średnim i niskim ryzyku. Czas skanowania wyniósł w przybliżeniu 15 minut. Wygenerowany raport jest czytelny i szczegółowy, jednak nie aż tak dokładny jak w przypadku OWSAP ZAP. Nie zawiera on jasnych informacji o podziale alertów na stopnie ryzyka. Podobnie jak w przypadku pierwszego narzędzia, raport zawiera opis problemu oraz dokładne miejsce wystąpienia wraz z wysłanym żądaniem, lecz nie posiada wskazówek rozwiązania podatności (Rys. 4.21). Jest również dostępna tabela podsumowująca wszystkie alerty podobnie jak w przypadku OWSAP ZAP (Rys 4.22) oraz lista podsumowująca wszystkie wykryte luki (Rys. 4.13). Smart Scanner poradził sobie najgorzej z wykrywaniem podatności, znalazł on najmniejszą liczbę alertów, w tym żadnego o wysokim ryzyku. Mimo, że sam skan trwał w przybliżeniu tylko 2 minuty, to jego dokładność była bardzo słaba. Raport generowany przez to narzędzie jest również mniej czytelny w porównaniu do poprzednich narzędzi, nie posiada on podsumowującej tabeli ze wszystkimi występującymi alertami (Rys 4.23).

- Wyniki w aplikacji do zarządzania projektami

W aplikacji stworzonej na potrzeby testów OWSAP ZAP wykrył 4 alerty o wysokim poziomie ryzyka co jest takim samym wynikiem jak Burp Suite. Oba narzędzia wykryły podatność wstrzykiwania kodu SQL oraz XSS w wielu miejscach (Rys. 4.12, Rys. 4.14). Pełne porównanie wykrytych luk zostało przedstawione w tabeli (Tabela 4.2). Podobnie jak w aplikacji WebGoat, także tutaj Smart Scanner poradził sobie najgorzej. Nie wykrył on żadnego miejsca wystąpienia wstrzykiwania kodu SQL, co jest najczęściej występującą podatnością w aplikacji. Czas trwania każdego skanu był podobny do poprzedniej aplikacji.

Podsumowując, zostały przeprowadzone pełne testy na obu aplikacjach oraz zostały wykryte wszystkie podatności oferowane przez aplikacje i narzędzia co umożliwia rzetelne porównanie ich efektywności w identyfikacji wspólnych podatności. Jednak nie wszystkie narzędzia były w stanie wykryć pełen zakres podatności obu aplikacji.

## SQL Injection (1)

▼ POST http://127.0.0.1:8080/WebGoat/SqlInjection/attack8

| | |
|---|---|
| **Alert tags** | ▪ OWASP_2021_A03<br>▪ CWE-89<br>▪ WSTG-v42-INPV-05<br>▪ OWASP_2017_A01 |
| **Alert description** | SQL injection may be possible. |
| **Other info** | The page results were successfully manipulated using the boolean conditions [a' AND '1'='1' -- ] and [a' OR '1'='1' -- ]<br><br>The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison.<br><br>Data was NOT returned for the original parameter.<br><br>The vulnerability was detected by successfully retrieving more data than originally returned, by manipulating the parameter. |
| **Request** | ▼ Request line and header section (796 bytes)<br><br>POST http://127.0.0.1:8080/WebGoat/SqlInjection/attack8 HTTP/1.1<br>host: 127.0.0.1:8080<br>Proxy-Connection: keep-alive<br>content-length: 46<br>sec-ch-ua-platform: "Windows"<br>X-Requested-With: XMLHttpRequest<br>User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36<br>Accept: */*<br>sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"<br>Content-Type: application/x-www-form-urlencoded; charset=UTF-8<br>sec-ch-ua-mobile: ?0<br>Origin: http://127.0.0.1:8080 |

**Rys. 4.17** Alarm wstrzykiwania kodu SQL OWSAP ZAP szczegóły 1[opracowanie własne]

```
host: 127.0.0.1:8080
Proxy-Connection: keep-alive
content-length: 46
sec-ch-ua-platform: "Windows"
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0
Safari/537.36
Accept: */*
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131",
"Not_A Brand";v="24"
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
sec-ch-ua-mobile: ?0
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?
username=adminuser
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: JSESSIONID=XSX-1KGNHe-
Z2ZzuUPGPcNcVLfCdMQuPL93KtJhk
```

▼ Request body (46 bytes)

```
name=a&auth_tan=a%27+AND+%271%27%3D%271%27+--+
```

**Rys. 4.18** Alarm wstrzykiwania kodu SQL OWSAP ZAP szczegóły 2[opracowanie własne]

| | |
|---|---|
| **Attack** | `a' OR '1'='1' --` |
| **Solution** | Do not trust client side input, even if there is client side validation in place.<br><br>In general, type check all data on the server side.<br><br>If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'<br><br>If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.<br><br>If database Stored Procedures can be used, use them.<br><br>Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!<br><br>Do not create dynamic SQL queries using simple string concatenation.<br><br>Escape all data received from the client.<br><br>Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input.<br><br>Apply the principle of least privilege by using the least privileged database user possible.<br><br>In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.<br><br>Grant the minimum database access that is necessary for the application. |

**Rys. 4.19** Alarm wstrzykiwania kodu SQL OWSAP ZAP szczegóły 3[opracowanie własne]

# Summaries

## Alert counts by risk and confidence

This table shows the number of alerts for each level of risk and confidence included in the report.

(The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.)

| | | Confidence | | | | |
|---|---|---|---|---|---|---|
| | | User Confirmed | Wysoki | Średni | Niski | Total |
| Risk | Wysoki | 0 (0,0%) | 0 (0,0%) | 3 (13,0%) | 1 (4,3%) | 4 (17,4%) |
| | Średni | 0 (0,0%) | 1 (4,3%) | 4 (17,4%) | 1 (4,3%) | 6 (26,1%) |
| | Niski | 0 (0,0%) | 0 (0,0%) | 4 (17,4%) | 2 (8,7%) | 6 (26,1%) |
| | Informacyjny | 0 (0,0%) | 2 (8,7%) | 4 (17,4%) | 1 (4,3%) | 7 (30,4%) |
| | Total | 0 (0,0%) | 3 (13,0%) | 15 (65,2%) | 5 (21,7%) | 23 (100%) |

**Rys. 4.20** Raport OWSAP ZAP podsumowanie alarmów[opracowanie własne]

1.1. http://127.0.0.1:8080/WebGoat/SqlInjection/attack8 [auth_tan parameter]

Next

## Summary

| | |
|---|---|
| Severity: | **High** |
| Confidence: | **Tentative** |
| Host: | **http://127.0.0.1:8080** |
| Path: | **/WebGoat/SqlInjection/attack8** |

## Issue detail

The **auth_tan** parameter appears to be vulnerable to SQL injection attacks. The payloads **53802550' or '6294'='6294** and **92012569' or '5046'='5049** were each submitted in the auth_tan parameter. These two requests resulted in different responses, indicating that the input is being incorporated into a SQL query in an unsafe way.

Note that automated difference-based tests for SQL injection flaws can often be unreliable and are prone to false positive results. You should manually review the reported requests and responses to confirm whether a vulnerability is actually present.

## Request 1

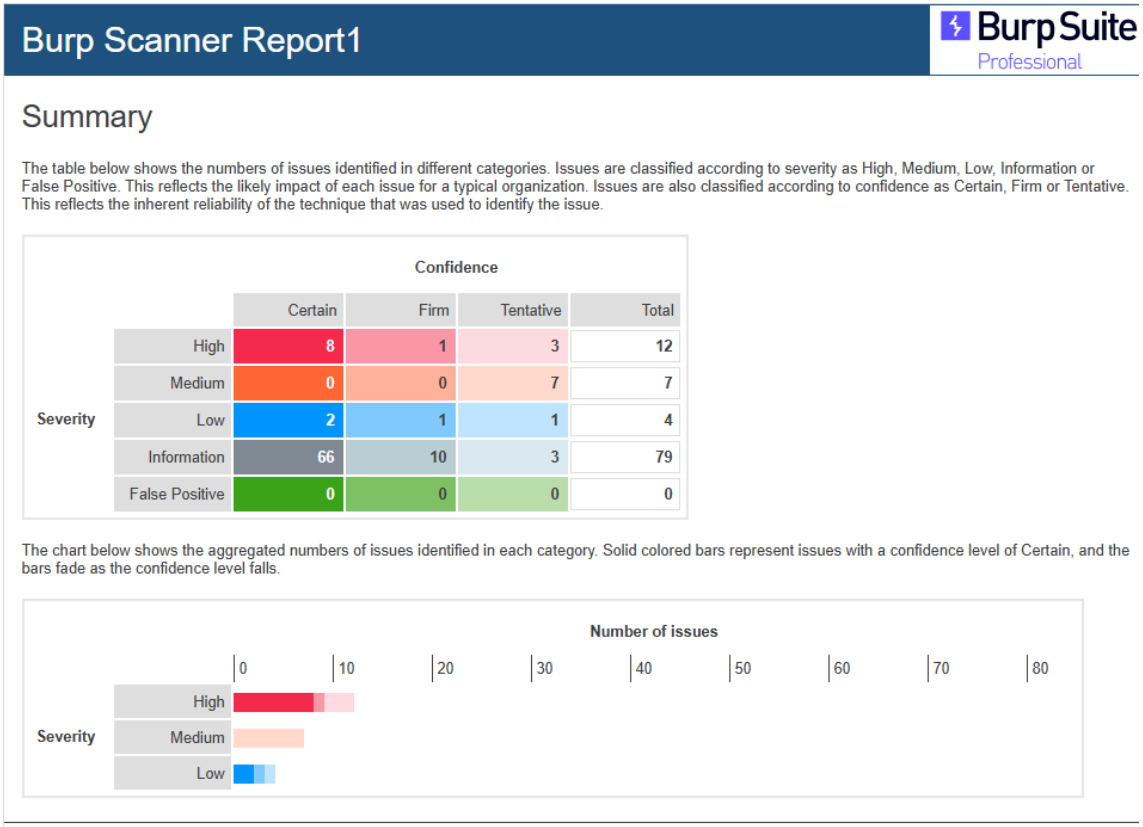
```
POST /WebGoat/SqlInjection/attack8 HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 21
sec-ch-ua-platform: "Windows"
Accept-Language: pl-PL,pl;q=0.9
sec-ch-ua: "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.140 Safari/537.36
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=diRSf5QmS0m5Vex7Th4MQiHJCrUaqulbqqr94ZFz
Connection: keep-alive
```

## Response 1

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Date: Thu, 16 Jan 2025 16:39:30 GMT
Content-Length: 1003

{
"lessonCompleted" : true,
"feedback" : "You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done
...[SNIP]...
```

**Rys. 4.21** Raport Burp Suite wstrzykiwania kodu SQL szczegóły[opracowanie własne]



Burp Scanner Report1

Burp Suite Professional

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low, Information or False Positive. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

| Severity | Confidence | | | |
|---|---|---|---|---|
| | Certain | Firm | Tentative | Total |
| High | 8 | 1 | 3 | 12 |
| Medium | 0 | 0 | 7 | 7 |
| Low | 2 | 1 | 1 | 4 |
| Information | 66 | 10 | 3 | 79 |
| False Positive | 0 | 0 | 0 | 0 |

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.



**Rys. 4.22** Raport Burp Suite podsumowanie alarmów[opracowanie własne]

## 3.1 No Redirection from HTTP to HTTPS

| | |
|---|---|
| SEVERITY | Medium |
| URL | http://127.0.0.1:8080/WebGoat/start.mvc?username=adminuser#lesson/SqlInjection.lesson/10 |

### DESCRIPTION

In scenarios where HTTPS is enabled but HTTP requests are not automatically redirected to HTTPS, users must explicitly use the HTTPS URL to ensure encrypted communication. Without redirection, HTTP traffic remains unencrypted and vulnerable to interception by attackers who can access the network interface.

### RECOMMENDATION

To enhance security, enforce the use of HTTPS by configuring your application or web server to redirect any HTTP request to HTTPS. Additionally, utilize the **Strict-Transport-Security** HTTP response header to provide an extra layer of security.

**Rys. 4.23** Raport Smart Scanner opis alarmu[opracowanie własne]

## *4.3 Identyfikacja mocnych i słabych stron poszczególnych rozwiązań*

Analiza narzędzi OWASP ZAP, Burp Suite, oraz Smart Scanner wykazała ich kluczowe mocne i słabe strony, co pozwala na precyzyjne określenie ich przydatności w różnych scenariuszach testowania bezpieczeństwa aplikacji internetowych.

### OWASP ZAP

**Mocne strony**:

- **Wszechstronność**: Wysoka skuteczność w wykrywaniu szerokiego zakresu podatności, w tym na przykład wstrzykiwanie kodu SQL, XSS oraz braków w nagłówkach bezpieczeństwa.
- **Przyjazny interfejs**: Jest prosty w obsłudze, szczególnie dla początkujących testerów.
- **Integracja z CI/CD**: Możliwość automatyzacji testów, co czyni go idealnym wyborem dla procesów DevSecOps.
- **Raportowanie**: Raporty wygenerowane są czytelne i szczegółowe z podziałem na poziomy ryzyka i linkami do dokumentacji OWASP Top Ten oraz z dokładnym żądaniem.
- **Otwarto źródłowy**: Darmowy dostęp i rozbudowana społeczność wspierająca rozwój narzędzia.

**Słabe strony**:

- **Czas skanowania**: Aktywne skanowanie zajmuje więcej czasu niż w innych narzędziach.

### Burp Suite

**Mocne strony**:

- **Zaawansowane funkcje**: Umożliwia manualne testowanie, co pozwala na szczegółową analizę nietypowych podatności.

- **Ekstensywność**: Możliwość rozbudowy poprzez wtyczki oraz integrację z innymi narzędziami.
- **Precyzja wykrywania**: Dostarcza szczegółowe informacje o podatnościach, co pomaga w zaawansowanych analizach.
- **Raportowanie:** Raporty są bardzo szczegółowe, zawierają dokładne miejsce luki oraz żądanie wysłane do wykrycia podatności wraz ze szczegółowym opisem problemu.
- **Szybkość działania**: Skanowanie jest bardzo szybkie w podstawowych scenariuszach. Jednak przy wyborze bardziej szczegółowego skanowania czas się wydłuża.

**Słabe strony**:

- **Trudność obsługi**: Interfejs i złożoność funkcji mogą być wyzwaniem dla mniej doświadczonych użytkowników.
- **Koszt**: Wersja profesionalna oferuje pełen zestaw funkcji, jednak jest płatna, co może być przeszkodą dla mniejszych firm i zespołów.

**Smart Scanner**

**Mocne strony**:

- **Szybkość działania**: Skany są wykonywane w krótkim czasie, co czyni go przydatnym w szybkich testach.
- **Prostota obsługi**: Intuicyjne narzędzie, łatwe do zrozumienia i użycia bez zaawansowanej wiedzy.

**Słabe strony**:

- **Niska skuteczność**: Wykrywa znacznie mniej podatności niż OWASP ZAP i Burp Suite, szczególnie tych o wysokim ryzyku.
- **Jakość raportów**: Raporty są mało czytelne i nie zawierają wystarczających szczegółów, co ogranicza ich przydatność w analizie.
- **Ograniczone funkcje**: Brak wsparcia dla bardziej zaawansowanych scenariuszy, takich jak testy API czy niestandardowych aplikacji.

# 5. Czynniki wpływające na skuteczność

## *5.1 Wpływ skomplikowania aplikacji na skuteczność narzędzi*

Skomplikowanie aplikacji webowych znacząco wpływa na efektywność narzędzi do testowania bezpieczeństwa. Złożoność programu może wynikać z wielu czynników. Na przykład z liczby funkcji oraz integracji z innymi systemami, użycia zaawansowanych technologii czy stopnia zaawansowania interfejsu użytkownika. Te elementy mogą sprawiać problemy dla narzędzi wykrywających podatności, które mogą mieć problemy z dokładnym skanowaniem.

OWSAP ZAP to narzędzie typu otwarto źródłowego, które oferuje duży zakres funkcji automatycznego skanowania. Jego skuteczność może jednak być ograniczona w przypadku bardzo złożonych aplikacji, w których treści są generowane dynamicznie. Skomplikowane mechanizmy uwierzytelniania mogą również utrudniać pełne pokrycie testowe. W takich sytuacjach konieczne jest ręczne dostosowanie konfiguracji narzędzia oraz uzupełnienie testów o manualne skanowanie.

Burp Suite jest znany ze swoich zaawansowanych i profesjonalnych funkcji, takich jak możliwość ręcznej manipulacji żądaniami i odpowiedziami HTTP. Radzi on sobie lepiej ze skomplikowanymi aplikacjami niż OWSAP ZAP. Posiada modułową budowę i możliwość integracji z różnymi rozszerzeniami, dzięki czemu pozwala na elastyczne dostosowanie do specyficznych wymagań aplikacji. Niemniej jednak, tester powinien posiadać wysoką wiedzę techniczną i doświadczenie w celu efektywnego wykorzystanie pełnego potencjału Burp Suite.

W przeciwieństwie do pozostałych narzędzi, Smart Scanner charakteryzuje się prostotą obsługi i szybkością działania. Może mieć jednak problemy w skutecznym testowaniu bardziej skomplikowanych aplikacji. Posiada ograniczone możliwości konfiguracyjne co sprawia, że może nie być stanie wykryć subtelnych lub specyficznych dla danej technologii podatności.

W miarę wzrostu złożoności aplikacji, automatyczne narzędzia do testowania bezpieczeństwa mogą napotykać na ograniczenia w identyfikacji wszystkich potencjalnych zagrożeń. Dlatego w takich przypadkach zaleca się łączenie automatycznych skanerów z

ręcznymi testami penetracyjnymi, aby zapewnić kompleksową ocenę bezpieczeństwa. Testerzy powinni być świadomi ograniczeń używanych narzędzi i dostosowywać strategie testowania do specyfiki i skomplikowania aplikacji, aby skutecznie identyfikować i neutralizować potencjalne zagrożenia.

### *5.2 Analiza adaptacyjności narzędzi do zmieniających się środowisk aplikacji*

W miarę z rozwijającym się światem aplikacji webowych, narzędzia do testowania bezpieczeństwa muszą być elastyczne i zdolne do dostosowywania się do nowych technologii, architektur oraz metodologii przez co adaptacyjność tych narzędzi determinuje ich skuteczność w identyfikowaniu podatności w różnorodnych i zmieniających się środowiskach aplikacyjnych.

OWASP ZAP jest narzędziem znanym ze swojej społeczności i regularnych aktualizacji, dzięki czemu potrafi szybko reagować na pojawiające się technologie i zagrożenia. Jego modularna architektura umożliwia integrację z różnymi środowiskami programistycznymi oraz procesami CI/CD, co czyni go wszechstronnym narzędziem w różnych kontekstach. Jednakże, w przypadku bardzo specyficznych lub niestandardowych technologii, może wymagać dodatkowej konfiguracji lub rozbudowy poprzez dostępne wtyczki. Burp Sui**te** za to oferuje szeroki zakres funkcji i rozszerzeń, które pozwalają na dostosowanie narzędzia do specyficznych potrzeb testera oraz charakterystyki testowanej aplikacji. Jego zdolność do integracji z różnymi środowiskami oraz wsparcie dla najnowszych technologii webowych sprawiają, że jest to narzędzie wysoko adaptacyjne. Dzięki regularnym aktualizacjom, Burp Suite pozostaje na bieżąco z najnowszymi trendami i zagrożeniami w obszarze bezpieczeństwa aplikacji. Smart Scanner, choć prostszy w obsłudze, może napotkać problemy w adaptacji do szybko zmieniających się środowisk aplikacyjnych. Jego mniejsza społeczność deweloperów sprawia, że aktualizacje mogą nie nadążać za najnowszymi technologiami i zagrożeniami. W efekcie, w przypadku aplikacji wykorzystujących nowoczesne rozwiązania technologiczne, Smart Scanner może nie zapewniać wystarczającej ochrony.

Podsumowując, OWASP ZAP i Burp Suite wykazują wysoką adaptacyjność do zmieniających się środowisk aplikacyjnych dzięki swojej elastyczności, możliwości rozbudowy oraz aktywnym społecznościom. Smart Scanner, mimo swojej intuicyjności, może nie sprostać wymaganiom dynamicznie rozwijających się technologii, co sprawia, że jego

przydatność jest mniejsza w bardziej zaawansowanych scenariuszach testowania bezpieczeństwa.

# 6. Wnioski

## *6.1 Podsumowanie głównych wyników analizy skuteczności narzędzi*

Cel niniejszej pracy został zrealizowany pomyślnie. Analiza skuteczności narzędzi OWSAP ZAP, Burp Suite oraz Smart Scanner wykazała różnorodność ich zastosowań oraz wyników w testach przeprowadzonych na aplikacji do zarządzania projektami oraz WebGoat. Najbardziej wszechstronnym narzędziem okazał się OWSAP ZAP, który wykrył największą liczbę podatności, szczególnie tych o wysokim ryzyku, na przykład wstrzykiwanie kodu SQL i XSS. Jest on szczególnie przyjazny dla początkujących testerów dzięki jego intuicyjnemu interfejsowi oraz łatwymi do czytania raportami z podziałem na poziomy ryzyka. Burp Suite z kolei dostarczał bardziej szczegółowych informacji o wykrytych podatnościach, co czyni go najlepszym narzędziem dla specjalistów bezpieczeństwa. Wydajność i szybkość działania były atutem Smart Scannera, który przeprowadzał testy najszybciej, ale wykrył niewielką liczbę podatności, głównie o niskim ryzyku, co ogranicza jego przydatność w bardziej wymagających scenariuszach. Raporty generowane przez to narzędzie były najmniej czytelne i brakowało w nich wskazówek dotyczących naprawy wykrytych problemów. W aplikacji WebGoat, narzędzia OWSAP ZAP i Burp Suite dostarczyły komplementarnych wyników, z przewagą OWSAP ZAP w zakresie liczby zidentyfikowanych podatności. W aplikacji do zarządzania projektami Burp Suite lepiej radził sobie z bardziej złożonymi zagrożeniami specyficznymi dla API.

Podsumowując, w podstawowych i średnio zaawansowanych testach bezpieczeństwa najlepiej sprawdził się OWSAP ZAP. Natomiast Burp Suite okazał się niezastąpiony w zaawansowanych analizach. Smart Scanner za to może być przydatny dla szybkiego skanowania, jednak jego ograniczona precyzja wykrywania i czytelność raportów czynią go gorszym narzędziem dla bardziej krytycznych aplikacji. Połączenie narzędzi daje najbardziej kompleksowe wyniki i pozwala na optymalne pokrycie szerokiego spektrum podatności.

## *6.2 Wskazówki dotyczące optymalnego wyboru narzędzi w różnych scenariuszach*

Istnieje wiele narzędzi do skanowania podatności aplikacji internetowych, ale wybór zależy od specyficznej charakterystyki środowiska oraz poziomu zaawansowania zespołu bezpieczeństwa. Kluczowe kryteria, jakie należy wziąć pod uwagę w wyborze narzędzi, obejmują rodzaj testowanej aplikacji, wymagania odnośnie raportowania oraz dostępne

zasoby. W przypadku prostych aplikacji lub podstawowych testów, narzędzie takie jak OWSAP ZAP posiada intuicyjnym, rozbudowany interfejs oraz skuteczne skanowanie popularnych podatności z listy OWSAP Top Ten. To czyni je odpowiednim wyborem dla początkujących użytkowników i małych zespołów. Burp Suite za to, szczególnie w wersji profesjonalnej, sprawdza się w bardziej skomplikowanych środowiskach, gdzie wymagana jest precyzja i ewentualna możliwość ręcznego testowania oraz integracja z procesami CI/CD. Dzięki prostocie obsługi i efektywności w wykrywania bardzo podstawowych luk, dla szybkich skanów środowisk produkcyjnych, Smart Scanner może być najbardziej użyteczny. Przy dużych projektach, gdzie aplikacje działają w środowiskach rozproszonych, najlepszym wyborem są narzędzia umożliwiające automatyzację i integrację z CI/CD, takie jak OWSAP ZAP, które pozwalają na ciągłe monitorowanie bezpieczeństwa. W kontekście aplikacji o wysokiej krytyczności (np. Systemy finansowe), warto postawić na narzędzia oferujące kompleksowe raportowanie i zaawansowaną analizę, jak Burp Suite. Podsumowując, optymalny wybór narzędzia powinien uwzględniać zarówno specyfikę projektu, jak i poziom zaawansowania użytkownika oraz możliwości integracyjne narzędzi. Warto również stosować kilka narzędzi jednocześnie, by zwiększyć szansę na wykrycie szerokiego zakresu podatności.

### *6.3 Propozycje dalszych badań w dziedzinie skanowania podatności*

Skanowanie podatności w aplikacjach webowych jest obszarem, który w ostatnich latach rozwija się bardzo szybko, co wiąże się z koniecznością adaptacji do nowych technologii, architektur oraz rosnących zagrożeń. Jeden z kierunków dalszych badań polega na integracja sztucznej inteligencji, która mogłaby służyć do przewidywania i klasyfikowania nowych podatności pod kątem wzorców historycznych. Jednym z ważniejszych obszarów jest również testowanie aplikacji bazujących na mikroserwisach lub chmurze. Architektura takich aplikacji jest bardziej skomplikowana co może wymagać nowych podejść do analizy bezpieczeństwa. Skanowanie nowoczesnych API, takich jak GraphQL to kolejne wyzwanie, gdyż stają się one coraz bardziej popularne wśród deweloperów. Integracja narzędzi z procesami CI/CD jest kolejnym ważnym aspektem przyszłych badań, ponieważ pozwala ona na automatyczne regularne testowanie w ramach DevSecOps. Nowe standardy bezpieczeństwa, takie jak kolejne wersje OWSAP Top Ten, blockchain czy IoT, nie tylko zwiększają stopień trudności w zakresie testowanie, ale również wymagają dostosowania istniejących narzędzi do odkrywania nowych typów zagrożeń. Wszystko to pokazuje jak

ważną rolę odgrywa potrzeba ciągłego doskonalenia i adaptacji narzędzi do skanowania podatności.

## Bibliografia


[1] OWSAP Top Ten 2021 Report, [Online]. Link: https://owasp.org/Top10/ (data dostępu 12.12.2024 r.).

[2] Suliman Alazimi, Daniel Conte de Leon, Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners, IEEE, 2019.

[3] Hewa Majeed Zangana, Exploring the Landscape of Website Vulnerability Scanners: A Comprehensive Review and Comparative Analysis, 2023.

[4] OWASP Foundation, OWASP ZAP Project [Online]. Link: https://owasp.org/www-project-zap/ (data dostępu 12.12.2024 r.).

[5] Pandey, S. Chaudhary, A., Vulnerability Scanning [Online]. Link: https://www.researchgate.net/publication/362137419_Vulnerability_Scanning

[6] Burp Suite Documentation, Burp Suite Professional Overview [Online]. Link: https://portswigger.net/burp/documentation/desktop (data dostępu 20.12.2024 r.).

[7] Arachni Project, Arachni Web Application Security Scanner[Online]. Link: https://www.arachni-scanner.com/ (data dostępu 19.12.2024 r.).

[8] Nirmal, K. Janet, B. Kumar, R, Web Application Vulnerabilities - The Hacker's Treasure, IEEE, 2018.

[9] Jaiswal, A. Raj,G. Singh, D, Security Testing of Web Applications: Issues and Challenges, IEEE, 2014.

[10] Stuttard, D., & Pinto, M., The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, ISBN, 2017.

[11] Gadepalli, S. (2020). Performance Evaluation of Web Vulnerability Scanners.

[12] Acunetix Documentation, [Online]. Link: https://www.acunetix.com/support/docs/introduction/ (data dostępu 18.12.2024 r.).

[13] OWSAP, Aplikacja WebGoat, [Online]. Link: https://github.com/WebGoat/WebGoat (data dostępu 05.12.2024 r.).

[14] Smart Scanner Documentation, [Online]. Link: https://www.thesmartscanner.com/docs/ (data dostępu 25.11.2024 r.).

[15] Rozporządzenie RODO, [Online]. Link: https://uodo.gov.pl/pl/404 (data dostępu 21.12.2024 r.).

[16] W3af Documentation, [Online]. Link: https://docs.w3af.org/en/latest/ (data dostępu 15.12.2024 r.).

[17] Common Vulnerabilities and Exposures. [Online]. Link: https://www.cve.org/ (dostęp 12.12.2024 r.).

## Spis Rysunków

## Spis Tabel