JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 37 (2025) 463–469
Received: 24 June 2025
Accepted: 20 September 2025

Analiza aktualnych zagrożeń i zabezpieczeń stosowanych w aplikacjach internetowych na przykładzie Symfony, Express i Spring Boot

Analysis of current threats and security measures used in web applications on the example of Symfony, Express, and Spring Boot

Magdalena Kramek*, Karol Mateusz Kurowski

Departament of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article analyzes the most common threats currently appearing in web applications and compares the built-in security features of Symfony, Express, and Spring Boot frameworks. The study aimed to identify security gaps, assess their risk, and then present practices that enable effective protection against attacks. The priority was to create four applications that were all identical in terms of structure. Tested applications were designed to have built-in security mechanisms from analyzed threats. The greatest threats currently are Broken Access Control attacks, cryptographic vulnerabilities, and code injection. The research process was conducted using Burp Suite Professional, SQLMap, XSSER, and Hydra tools. The results indicate that Symfony and Spring Boot are the best protected against the threats. Additionally, the default Express skeleton mechanisms do not protect the application from Cross Site Scripting (XSS) attacks.

Keywords: security; Symfony; Express; Spring Boot

Streszczenie

Artykuł obejmuje analizę najczęstszych zagrożeń pojawiających się obecnie w aplikacjach internetowych oraz porównanie zabezpieczeń wbudowanych w szkielety programistyczne Symfony, Express oraz Spring Boot. Badanie miało na celu zidentyfikowanie luk w bezpieczeństwie, ocenę ich ryzyka, a następnie przedstawienie praktyk umożliwiających skuteczną ochronę przed atakami. Priorytetem było stworzenie czterech aplikacji, które będą identyczne pod względem swojej struktury. Testowane aplikacje zostały zaprojektowane w taki sposób, aby posiadały wbudowane zabezpieczenia przeciw analizowanym zagrożeniom. Największe zagrożenie obecnie stanowią ataki typu Broken Access Control, luki kryptograficzne i wstrzykiwanie kodu. Proces badawczy został przeprowadzony z wykorzystaniem narzędzi Burp Suite Professional, SQLMap, XSSER oraz Hydra. Otrzymane wyniki wskazują, że najlepiej zabezpieczonymi pod względem wyżej wymienionych zagrożeń są Symfony oraz Spring Boot. Ponadto domyślne mechanizmy szkieletu Express nie zabezpieczają aplikacji przed atakami Cross Site Scripting (XSS).

Słowa kluczowe: bezpieczeństwo; Symfony; Express; Spring Boot

*Corresponding author

E-mail address: s95453@pollub.edu.pl (M. Kramek)

1. Wstęp

W dobie szerokiego rozwoju technologicznego społeczeństwo staje się coraz bardziej zależne od systemów internetowych. Te stanowią już nieodłączną część ludzkiego życia. Aplikacje upraszczają codzienne życie, umożliwiają ludziom korzystanie z bankowości internetowej. Dzięki ich rozwojowi strony rządowe pozwalają na składanie dokumentów urzędowych poprzez internet. Obecność mediów społecznościowych oraz platform streamingowych, które w swoich zamierzeniach mają dostarczać rozrywki, ostatecznie pochłania ogromną ilość czasu każdego człowieka.

Wszystkie te aplikacje z pewnością potrafią wiele ułatwić, jednak należy pamiętać, że ze względu na ich sposób działania wymuszają na użytkowniku podawanie jego prywatnych danych. By mógł on w pełni korzystać z zapewnianych przez nie funkcjonalności, systemy zapisują wiele informacji. Wiąże się to z nieodłącznym ryzykiem utraty wrażliwych danych. Niebezpieczeństw dotyczących korzystania z aplikacji internetowych jednak jest z dnia na dzień coraz więcej, do czego przyczynia się m.in. powszechność złośliwych systemów czy ograniczona wiedza użytkowników dotyczących bezpieczeństwa. W świetle tych zagrożeń bardzo ważna jest kwestia zabezpieczeń wbudowanych w aplikacje internetowe.

Implementowanie własnych zabezpieczeń przez zespoły programistyczne potrafi być czasochłonne i wymagać specjalistycznej wiedzy z zakresu cyberbezpieczeństwa. Natomiast wiele współczesnych szkieletów programistycznych udostępnia gotowe moduły, które mogą pomóc ochronić aplikację na wielu płaszczyznach. Jednak w zależności od technologii i sposobu implementacji dane szkielety mogą charakteryzować się różnym poziomem odporności.

Dostępne były nieliczne źródła literatury analizujące bezpieczeństwo poszczególnych szkieletów [2, 3, 4], jednak nie zostało do tej pory przeprowadzone szczegółowe badania dotyczące zabezpieczeń w szkieletach Symfony, Express oraz Spring Boot. Odnalezione pozycje jedynie pobieżnie przedstawiały kwestie bezpieczeństwa, głównie skupiając się na wydajności poszczególnych szkieletów. Niniejszy artykuł właśnie tym się będzie wyróżniać, że bezpieczeństwo wskazanych szkieletów zostało

przetestowane i zweryfikowane. Ze względu na wynik analizy źródeł zdecydowano się połączyć aspekt badań zagrożeń z wbudowanymi zabezpieczeniami, by następnie na ich przykładach zebrać cenne wskazówki dotyczące pracy z danym szkieletem programistycznym.

2. Przegląd literatury

W celu przeprowadzenia wartościowej analizy aktualnych zagrożeń, a następnie wykonania szczegółowych testów bezpieczeństwa wybranych szkieletów programistycznych zapoznano się z dotychczasowymi pozycjami literatury. Kluczowym źródłem okazał się raport stworzony przez fundację OWASP (Open Worldwide Application Security Project) [https://owasp.org], który przedstawia ranking dziesięciu najczęstszych zagrożeń w aplikacjach internetowych. Na szczycie klasyfikacji [1] znajdują się ataki z kategorii typu Broken Access Control dotyczące nieautoryzowanych dostępów do zasobów systemu. Obejmuje ona m.in. pozyskiwanie dostępu do wrażliwych informacji, wprowadzanie modyfikacji czy nawet usuwanie danych lub przeprowadzanie działań poza zasięgiem dedykowanego użytkownika. Do drugich w kolejności zagrożeń należą luki kryptograficzne, które grupują przestarzałe algorytmy mieszające, domyślne lub zbyt krótkie klucze szyfrujące czy też używanie losowości do celów kryptograficznych. Na trzeciej pozycji widnieje wstrzykiwanie, które opiera się na przekazywaniu np. w polach formularza kodu wydobywającego rekordy z bazy danych. W przypadku braku filtrowania pozyskiwanych ciągów znaków taka aplikacja nie zapewnia odpowiedniego bezpieczeństwa danym.

W konferencji Critical Infrastructure Protection in the Light of the Armed Conflicts znajduje się artykuł [2], w którym Zlatko Čović dokonał opisu wyżej wymienionych zagrożeń bezpieczeństwa aplikacji webowych. Na podstawie przykładów przedstawia scenariusze ataków oraz oferowane rozwiązania ochrony przed tymi zagrożeniami, podkreślając znaczenie bezpieczeństwa w codziennym korzystaniu z aplikacji webowych.

W artykule z 2018 roku [3] zaprezentowano badanie podatności na wstrzykiwanie kodu SQL, wstrzykiwanie kodu JavaScript – XSS (ang. Cross Site Scripting) oraz na ataki CSRF (ang. Cross-Site Request Forgery). Wspomniane ataki polegają na fałszowaniu określonych zapytań i przekazywaniu ich klientowi w celu uzyskaniu dostępu do autoryzowanych zasobów aplikacji. Autorzy podkreślają znaczenie wiedzy w zakresie bezpieczeństwa i zalecają, by programiści zapoznawali się z wytycznymi, podatnościami oraz sposobami, w jaki można zabezpieczać zasoby na poziomie pisania kodu. W zakresie analizy zagrożeń bezpieczeństwa w systemach elektronicznej dokumentacji medycznej EMR (ang. Electronic Medical Record) zbadano podatności na ataki takie jak XSS, wstrzykiwanie kodu SQL, ataki CSRF oraz niewystarczająco bezpieczne uwierzytelnianie [4]. Porównanie polegało na zestawieniu funkcji bezpieczeństwa trzech popularnych frameworków języka PHP: Laravel, CodeIgniter i Symfony. Wyniki wskazują szkielet programistyczny Laravel jako oferujący najbardziej wszechstronne zabezpieczenia wymagane w przypadku tego rodzaju systemów. Dane, które przechowują, są szczególnie wrażliwe ze względu na ich charakter, dlatego wskazane podatności mają ogromne znaczenie w kontekście zabezpieczania aplikacji webowych.

W [5] autor dokonał porównania trzech szkieletów serwerowych ASP.NET MVC 5, Symfony dla PHP oraz Node.js Express m.in. w zakresie bezpieczeństwa, jednak wykazuje ono zaledwie podobieństwo polityki bezpieczeństwa ASP.NET MVC z tą dotyczącą Symfony. We wnioskach znalazły się rodzaje aplikacji pod względem ich dopasowania do poszczególnych szkieletów.

W 2023 pojawił się artykuł [6], w którym autorzy wzięli pod uwagę projekty typu open-source do analizy bezpieczeństwa w językach Java oraz PHP. Wśród testowanych szkieletów programistycznych znalazły się: dla Javy Spring, Play, Spark, Vaadin, Vert.x-Web oraz dla PHP Symfony, CakePHP, Slim, Laravel, Zend/Laminas. Badanie wykazało, że jako te najpopularniejsze, Laravel oraz Spring pod względem bezpieczeństwa wypadły słabo w porównaniu z Vert.x-Web ze znacznie mniejszą liczbą użytkowników. We wnioskach podkreślono znaczenie poszerzania własnej wiedzy w zakresie bezpieczeństwa aplikacji, ponieważ złożone systemy wymagają większej ochrony, której wbudowane zabezpieczenia mogą nie zapewniać. Ponadto niezwykle ważnym jest pełne zrozumienie sposobu działania danego szkieletu programistycznego, ponieważ programiści nie mogą całkowicie ufać wbudowanym filtrom.

3. Cel oraz hipotezy badawcze

Celem niniejszej pracy było wyodrębnienie oraz analiza obecnych zagrożeń dotyczących aplikacji internetowych, co miało umożliwić porównanie wbudowanych zabezpieczeń w szkieletach programistycznych pod względem tych najczęściej występujących zagrożeń. Badanie miało na celu identyfikację luk w bezpieczeństwie, ocenę ryzyka oraz przedstawienie praktyk na przykładach wspomnianych technologii. Po zidentyfikowaniu zagrożeń została przeprowadzona analiza, której wynikiem jest rekomendacja umożliwiająca wybór najbardziej odpowiedniej technologii pod względem zabezpieczeń.

Na potrzeby badań, opierając się na przeglądzie literatury, zostały sformułowane następujące hipotezy:
H1: Najgroźniejszymi atakami na aplikacje internetowe są te związane z Broken Access Control.
H2: Wykorzystanie frameworków internetowych pomaga zwiększyć bezpieczeństwo aplikacji.
H3: Frameworki internetowe oferują podobny poziom zabezpieczeń, który zależy od sposobu i poprawności ich implementacji.

4. Metodyka badań

Badanie rozpoczęto od przeprowadzenia analizy zagrożeń, podczas której wyłoniono potencjalnie najbardziej niebezpieczne podatności aplikacji internetowych.

4.1. Środowisko badawcze

Część praktyczną eksperymentu przygotowano w oparciu o trzy aplikacje internetowe, każdą z nich napisano w innym języku programowania. Ich kluczową

funkcjonalność stanowiła możliwość utworzenia prostego blogu internetowego, gdzie użytkownicy mogliby dodawać, edytować oraz usuwać własne posty. Każda z tych aplikacji została zaimplementowana według następujących kryteriów:

- aplikacja musi być rodzaju MVC (ang. Model-View-Controller),
- aplikacja powinna wykorzystywać silniki szablonów właściwe dla danej technologii,
- struktura bazy danych musi być analogiczna dla wszystkich aplikacji,
- aplikacja powinna posiadać zaimplementowane moduły CRUD (ang. Create Read Update Delete) dla encji User oraz Post,
- aplikacja powinna posiadać zaimplementowany mechanizm autoryzacji i autentykacji,
- część modułów aplikacji powinno być dostępnych po uprzednim zalogowaniu przez użytkownika,
- aplikacja powinna posiadać dwa punkty w REST API wykorzystujące metody GET i POST,
- aplikacja powinna stosować wbudowane mechanizmy bezpieczeństwa właściwe dla danego szkieletu zgodnie z jego dokumentacją.

Zapewnienie jednakowej funkcjonalności i złożoności było konieczne, aby zapewnić miarodajność testów i umożliwić zestawienie wyników ze sobą.

Ponadto w celach porównawczych została przygotowana aplikacja internetowa w czystym języku PHP, której funkcjonalności pozostały bez zmian, natomiast pozbawiono ją zabezpieczeń. Utworzenie wspomnianego systemu było konieczne, aby w klarowny sposób przedstawić skalę błędów i podatności, jakie miałyby miejsce, gdyby nie korzystano z żadnych mechanizmów bezpieczeństwa. Technologie użyte podczas badań przedstawione zostały w Tabeli 1.

Tabela 1: Technologie użyte do implementacji aplikacji

Wersja	Nazwa szkieletu	Wersja szkieletu	Baza danych	Badane moduły
Java 21	Spring Boot	3.4.1	PSQL	Spring Security
PHP 8.3	Symfony	7.2	PSQL	Symfony Security
Node.js 23.5.0	Express.js	4.21.2	PSQL	Biblioteki zewnętrzne
PHP 8.3	Brak	Brak	PSQL	Brak

Do implementacji użyto najnowszych stabilnych wersji szkieletów (tj. kwiecień 2025). Narzędzia do testów opisano w podrozdziale 4.2. Do wykonania testów użyto platformę testową, tj. komputer stacjonarny, którego szczegółową specyfikację umieszczono w Tabeli 2. Do uruchamiania aplikacji wykorzystane były domyślne środowiska dostarczane razem ze szkieletami, czyli dla Spring użyto serwera TomCat, dla Symfony i Express był to wbudowany serwer, dla PHP serwer Apache. Do połączenia użyto protokołu HTTPs z wygenerowanymi domyślnymi certyfikatami. Dzięki temu wyeliminowane zostały problemy związane z transmisją danych.

Tabela 2: Platforma testowa

Komponent	Specyfikacja
Procesor	AMD RYZEN 7 9700X
Rozmiar pamięci RAM	32 GB DDRM5
Dysk twardy	SSD PCIE 2TB
System operacyjny	Windows 11 + Kali Linux

Trudnością związaną z testowaniem bezpieczeństwa stanowił dobór wymiernych metryk, na podstawie których można by było zestawić ze sobą wyniki przebadanych aplikacji. Podczas analizy zagrożeń wyłonione zostały obecne zagrożenia, które można było przetestować w sposób powtarzalny i niezależny od człowieka:

- ataki związane z Broken Access Control, czyli nieautoryzowany dostęp do stron,
- ataki SQL Injection związane z wstrzykiwaniem złośliwego kodu SQL,
- ataki XSS polegające na wstrzykiwaniu złośliwego kodu JavaScript.

4.2. Narzędzia

By przetestować podatności aplikacji na wymienione zagrożenia należało wyznaczyć odpowiednie oprogramowanie automatyzujące testy dla każdej kategorii osobno. Aplikacje zostały przetestowane pod kątem bezpieczeństwa w sposób ogólny przy pomocy systemu Burp Suite Professional. Program ten pomógł w utworzeniu raportu dotyczącego błędów, podzielonych według stopnia niebezpieczeństwa. Początkowo w tym celu planowano wykorzystać program OWASP ZAP, jednak podczas testów próbnych zwrócił dużą ilość fałszywie negatywnych wyników, co wymusiło zmiany narzędzia. Do szczegółowych testów podatności każdego z trzech najpilniejszych zagrożeń przydzielono następujące narzędzia:

- Hydra umożliwiająca łamanie haseł poprzez ataki słownikowe i Brute Force,
- SQLMap pozwalający wykonywać wstrzykiwanie złośliwego kodu SQL do bazy danych w celu wykradania danych lub ich usuwania dla ataków SQL Injection,
- XSSER dający możliwość wstrzykiwania kodu JS lub HTML w punkty aplikacji dla ataków XSS.

4.3. Scenariusz testowy

Aplikacje zostały przetestowane w następujący sposób.
Etap 1: Ogólne testy bezpieczeństwa z wykorzystaniem programu Burp Suite Professional, gdzie wynikiem testu była lista błędów podzielonych na kategorie podatności.
Etap 2: Testy na podatność SQL Injection przy pomocy programu SQL MAP. Test polegał na ataku pięciu wybranych miejsc strony internetowej: logowanie (1), rejestracja (2), dodanie postu (3), edycja postu (4) oraz edycja użytkownika (5). Wynikiem testów była liczba wykonanych ataków oraz znalezionych podatności.
Etap 3: Testy na podatność XSS przy użyciu programu XSSER. Badanie przeprowadzone zostało analogicznie do testów SQL Injection z identycznym strukturą zwracanych wyników.

Etap 4: Testy podatności na złamanie kontroli dostępu za pomocą narzędzia Hydra. Wynikiem było zestawienie liczby prób złamania dostępu do liczby skutecznych włamań do aplikacji. Testy te przeprowadzono z podziałem na dwie kategorie:

1. Wykorzystanie techniki Brute Force, gdzie Hydra próbuje "zgadnąć hasło".
2. Wykorzystaniem techniki słowników z popularnymi hasłami dostępnymi w zbiorach danych.

Wyniki testów zostały opracowane i w miarę możliwości przekonwertowane do postaci liczbowej, a następnie przedstawione w Tabelach. W rozdziale dotyczącym analizy wyników omówione zostały również pozostałe cechy szkieletów wpływające na bezpieczeństwo wraz z analizą podatności szczególnych dla danych technologii. Dodatkowo zamieszczono wnioski i oceny implementacji aplikacji oraz ich zabezpieczeń.

5. Wyniki

5.1. Testy automatyczne

Wyniki pierwszego etapu badań, który zakładał testy automatyczne bezpieczeństwa aplikacji przedstawiono na Rysunkach 1 - 4. Tabele przedstawiają dane z raportów z Burp. Problemy zostały sklasyfikowane według statusu jako *Wysoki* (kolor czerwony), *Średni* (kolor pomarańczowy), *Niski* (kolor niebieski), *Informacje* (kolor szary) lub *Fałszywie Pozytywne* (kolor zielony). Odzwierciedlają one prawdopodobny wpływ każdej kwestii na typową organizację. Dodatkowa klasyfikacja przebiega według pewności problemów: *Pewne, Silne* lub *Niepewne* (w tabelach rozróżniane nasyceniem koloru dla danego statusu), które z kolei odpowiadają wiarygodności techniki wykorzystanej do identyfikacji problemu.

Rysunek 1 przedstawia wyniki testów aplikacji kontrolnej, która nie posiadała zaimplementowanych zabezpieczeń. Pomimo niskiego stopnia złożoności aplikacji, znaleziono aż 37 problemów o wysokim statusie. Były to głównie miejsca wrażliwe na ataki XSS i SQL Injection (po 15 problemów). Dodatkowo wykryty został problem z CSD (ang. Client-side Desync). Jest to sytuacja, w której dane po stronie klienta stają się niespójne z rzeczywistym stanem utrzymywanym przez serwer i umożliwia atakującemu ominięcie walidacji lub manipulacje danymi. Ponadto znalezione zostały problemy z CSRF (ang. Cross-site Request Forgery) oraz braki szyfrowania danych po stronie serwera.

		Pewność			
		Pewne	Silne	Niepewne	Razem
Status	Wysoki	34	0	3	37
	Średni	0	1	1	2
	Niski	1	1	0	2
	Informacyjne	32	8	4	44
	Fałszywie pozytywne	0	0	0	0

Rysunek 1: Raport dla aplikacji PHP.

Na Rysunku 2 można zauważyć wyraźnie spadek liczby błędów, jednak warto odnotować, że nadal wystąpiły problemy z wysokim statusem. Znalezione zostały podatności na ataki XSS. Dodatkowo ponownie wystąpiły problemy z szyfrowaniem danych (1) oraz braki CSRF (2).

		Pewność			
		Pewne	Silne	Niepewne	Razem
Status	Wysoki	3	0	2	5
	Średni	0	0	0	0
	Niski	1	0	0	1
	Informacyjne	22	8	1	31
	Fałszywie pozytywne	0	0	0	0

Rysunek 2: Raport dla aplikacji Express.

Na Rysunkach 3 i 4 zawierających wyniki kolejno dla Symfony i Spring widoczne jest brak błędów o statusie wysokim. Jedyne błędy zaklasyfikowane jako *poważniejsze* to te związane z niezweryfikowanym certyfikatem TLS, jednak problem ten dotyczy już właściwych serwerów niż samych aplikacji. Dodatkowo dla szkieletu programistycznego Symfony został wykryty problem z możliwym atakiem typu SSL stripping, umożliwiający komunikację z aplikacją przy wykorzystaniu niezabezpieczonego protokołu HTTP.

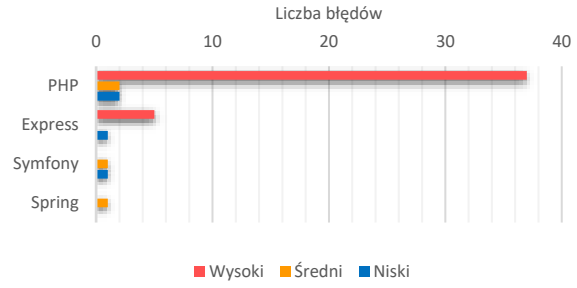
		Pewność			
		Pewne	Silne	Niepewne	Razem
Status	Wysoki	0	0	0	0
	Średni	1	0	0	1
	Niski	1	0	0	1
	Informacyjne	23	14	1	38
	Fałszywie pozytywne	0	0	0	0

Rysunek 3: Raport dla aplikacji Symfony.

		Pewność			
		Pewne	Silne	Niepewne	Razem
Status	Wysoki	0	0	0	0
	Średni	1	0	0	1
	Niski	0	0	0	0
	Informacyjne	44	0	0	44
	Fałszywie pozytywne	0	0	0	0

Rysunek 4: Raport dla aplikacji Spring.

Na Rysunku 5, zaprezentowano zestawienie sumarycznej liczby problemów dla poszczególnych technologii.

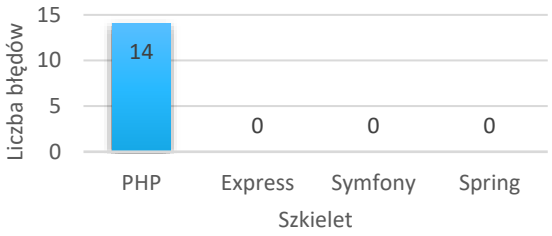


Rysunek 5: Porównanie liczby znalezionych problemów z podziałem na framework.

5.2. Testy szczegółowe

W drugim etapie badań przeprowadzone zostały testy sprawdzające odporność aplikacji na konkretne ataki. Na Rysunku 6 zostały zaprezentowane wyniki sumaryczne ilości znalezionych podatności SQL Injection z programu

SQLMap dla wszystkich 5 punktów końcowych. Należy wspomnieć, że w aplikacji kontrolnej PHP każdy z punktów końcowych posiadał podatność na atak, a liczba znalezionych luk wynosiła od 2 do 3 na każde badany element aplikacji.



Rysunek 6: Wyniki testu z programu SQLMap.

Dla danych zebranych z narzędzia XSSER w etapie trzecim, został wyliczony procent skutecznych ataków według wzoru:

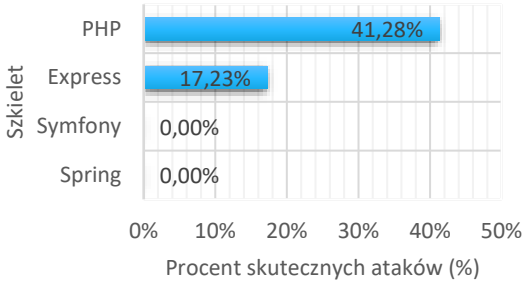
$$AttackEffectiveness = \frac{\sum_{i=1}^{5} S_i}{\sum_{i=1}^{5} P_i} \times 100\% \qquad (1)$$

gdzie S_i jest liczbą skutecznych ataków na i punkcie końcowym, P_i jest całkowitą liczbą ataków na i punkt końcowy.

Szczegółowe dane zostały zaprezentowane w Tabeli 3. Wyniki końcowe są zaprezentowane na Rysunku 7.

Tabela 3. Szczegółowe wyniki z programu XSSER

	PHP		Express		Symfony		Spring	
Nr	(Pi)	(Si)	(Pi)	(Si)	(Pi)	(Si)	(Pi)	(Si)
1	2582	0	2582	0	2582	0	2582	0
2	1291	0	3872	0	3872	0	3872	0
3	2582	1199	2582	104	2582	0	3873	0
4	2582	2346	3873	2565	2582	0	3873	0
5	2582	1251	2582	0	3873	0	3873	0



Rysunek 7. Wyniki testu z programu XSSER.

Ostatni etap badań miał na celu sprawdzenie zabezpieczenia logowania na ataki słownikowe (ang. Dictionary Attack DA) oraz typu Brute Force (BF). Liczbą kombinacji ataku DA, jest długość słownika wykorzystywanego do ataku. Do oszacowania wartości dla ataku BF wykorzystano wzór dla liczby wariancji z powtórzeniami. Długość hasła była stała i równa 6. W pierwszym przypadku dla haseł złożonych z małych liter alfabetu łacińskiego liczba kombinacji wynosi 26^6, czyli 3,09E+08. Drugi testowany przypadek zakłada wykorzystanie małych i dużych liter oraz cyfr, dając łącznie 62^6, czyli 5,68E+10 możliwych wartości. Szacowany, pesymistyczny czas złamania hasła został pobrany z programu Hydra, który wylicza go na podstawie średniej liczby prób ataku na minutę. Końcowe wyniki zostały zaprezentowane w Tabeli 4.

Tabela 4. Szczegółowe wyniki z programu Hydra

Liczba kombinacji	Metoda	Długość	Czas (h)			
			PHP	Express	Symfony	Spring
1,43E+07	DA	5	20	3700	3700	3700
3,09E+08	BF	6	470	80446	80446	80446
5,68E+10	BF	6	8,1E+04	1,5E+07	1,5E+07	1,5E+07

6. Analiza wyników

6.1. Testy automatyczne

Zbiorcze dane dla testów automatycznych przedstawione na Rysunku 5, pokazują, że wykorzystanie szkieletów programistycznych zdecydowanie obniżyło liczbę błędów aplikacji. Express poprawił bezpieczeństwo aplikacji, ale nie zabezpieczył on aplikacji przed atakami XSS tak, jak miało to miejsce dla Symfony i Spring Boot. Było to prawdopodobnie spowodowane tym, że silnik szablonów oraz ORM nie czyściły wyników zwracanych, co powodowało to, że skrypt był przekazywany w niezmienionej formie, co umożliwiało poprawne interpretowanie go przez przeglądarkę. Stanowi to bardzo poważną luką bezpieczeństwa. Symfony i Spring Boot wykazały się dobrym poziomem bezpieczeństwa, nie znaleziono w nich błędów. Jedynymi problemami były niezweryfikowane certyfikaty TLS.

6.2. Analiza wyników z programu SQLMAP

Do przeprowadzenia analizy konieczne było wyliczenie procentu skutecznych ataków w analogiczny sposób do tego, które zostało zaprezentowane w sekcji 5.2. Otrzymane w ten sposób wartości średnie zostały poddane testowi Shapiro-Wilka [7] w celu weryfikacji, czy pochodzą one z rozkładu normalnego. Tylko zbiór danych próbki kontrolnej wykazał taką właściwość, więc należało w następnym kroku skorzystać z testu nieparametrycznego Kruskala-Wallisa, umożliwiającego wskazania różnic pomiędzy przynajmniej dwiema grupami. W przeprowadzonym badaniu przyjęto hipotezę zerową, zakładającą brak istotnych różnic w skuteczności ataków między poszczególnymi aplikacjami. Natomiast hipoteza alternatywna dopuszczała istnienie takich różnic przynajmniej pomiędzy jedną parą porównywanych grup. Test ten wykazał istotność statystyczną, ponieważ p wyniosło 0.00035, co jest mniejsze od przyjętego progu $p < 0.05$ i oznacza odrzucenie hipotezy standardowej na rzecz alternatywnej. W celu identyfikacji istotnych par różniących się statystycznie użyto testu Dunna z poprawką Holma, [8] który wykorzystywany jest jako test post-hoc. Wykazał on, że próbka kontrolna PHP

wykazuje istotnie wyższą skuteczność ataków w porównaniu do Express (p = 0.0027), Spring (p = 0.0022) i Symfony (p = 0.0018), natomiast między Express, Spring i Symfony nie stwierdzono istotnych różnic.

6.3. Analiza wyników z programu XSSER

Analiza danych zebranych w tym badaniu została przeprowadzona analogicznie do wyników z programu SQLMap, z tą różnicą, że wyniki zostały ograniczone do punktów końcowych, na których była możliwość skutecznego przeprowadzenia ataku XSS, ponieważ niektóre z nich. jak np. login, nie wykazały podatności nawet dla próbki kontrolnej. Podczas testu Shapiro-Wilka ponownie została odrzucona hipoteza normalności zbiorów, więc ponownie przeprowadzono test Kruskala-Wallisa. Otrzymany wynik wskazuje, że istnieją statystycznie istotne różnice w skuteczności ataków pomiędzy co najmniej dwoma z aplikacji (p = 0.04216 < 0.05). Wyniki testu Dunna pokazują, że po uwzględnieniu korekty Holm żadna z par grup nie różni się statystycznie istotnie (wszystkie p > 0.05). Najniższe p-wartości skorygowane dotyczą porównań PHP vs Spring (p = 0.0986) i PHP vs Symfony (p = 0.0822), co sugeruje tendencję do różnic, ale nie przekraczającą poziomu istotności. Chociaż test Kruskala-Wallisa wskazał ogólną różnicę między grupami (p = 0.04216), test Dunna nie potwierdził jednoznacznie istotnych różnic między parami. Może to oznaczać, że różnice są subtelne lub próbka jest zbyt mała, by wykazać istotność w testach parami.

6.4. Analiza wyników z Hydra

Analiza wykazała, że badane szkielety pozwalają na zabezpieczenie aplikacji przed atakami na łamanie haseł. Możliwość konfigurowania ilości prób i czasu oczekiwania na ponowne wpisanie hasła, znacznie wydłuża czas włamania. W testach korzystano ze standardowego ustawienia oznaczającego 5 prób co 5 minut. Hydra przy wykorzystaniu 64 wątków musiała zostać ograniczona do wysyłania 5 zapytań co 5 minut, co średnio dawało zaledwie 64 próby na minutę. Dla porównania aplikacja kontrolna umożliwiała wykonanie do 11000 prób na minutę.

7. Podsumowanie

Otrzymane wyniki umożliwiają weryfikację przyjętych hipotez badawczych. Hipoteza numer jeden dotycząca najgroźniejszych ataków na aplikacje internetowe związanych z BAC została potwierdzona. Dowodzi tego przegląd literatury a w szczególności raport OWASP, gdzie problem ten znajduje się na szczycie rankingu.

Podobnie jak hipoteza numer dwa o zwiększeniu bezpieczeństwa aplikacji internetowych poprzez wykorzystanie szkieletów. Analiza porównawcza wykazała zniwelowanie błędów bezpieczeństwa w opartych o nie aplikacjach w porównaniu do próbki kontrolnej.

Natomiast ostatnia hipoteza została odrzucona, ponieważ szkielet programistyczny Express jako jedyny z badanych nie oferował podobnego poziomu zabezpieczeń, nie zabezpieczając aplikacji przed atakami XSS.

Pośród badanych szkieletów programistycznych najlepiej zabezpieczonymi okazały się Symfony oraz Spring Boot. Oba szkielety udostępniają w opinii autorów dosyć dobre mechanizmy wbudowane chroniące bezpieczeństwo aplikacji. Należy docenić również prostotę ich implementacji oraz duże możliwości konfiguracyjne danych komponentów. Programiści mogą korzystać z gotowych rozwiązań typu mechanizmy uwierzytelnienia i autoryzacji, wsparcie wielu algorytmów szyfrujących czy wbudowane w formularze tokeny CSRF. Ponadto umożliwiają również łatwe dołączanie dodatkowych funkcji zwiększających bezpieczeństwo, takich jak logowanie dwuetapowe czy kontrola logów aplikacji. Umożliwiają także zastosowanie mechanizmu soft-delete, czyli nietrwałego usuwania danych z bazy, co pozwala na ich proste odzyskiwanie. Dodatkowo, mechanizm głosujących daje możliwość szczegółowego konfigurowania dostępu do poszczególnych modułów aplikacji. Decyzje o przyznaniu dostępu mogą być podejmowane na podstawie ocen wielu niezależnych komponentów lub klas tzw. – głosujących, które wspólnie określają, czy użytkownik powinien uzyskać dostęp. Dzięki temu zmniejsza się ryzyko fałszywie pozytywnej autoryzacji (np. zalogowania się nieuprawnionej osoby) i wzmacnia ochronę przed atakami BAC.

Express natomiast, który w założeniu ma być prostym i szybkim szkieletem nie udostępnia jednego gotowego komponentu, tylko trzeba polegać na wielu zewnętrznych bibliotekach od różnych dostawców oraz nie wspiera tak dobrej integracji z silnikami szablonów jak ma to miejsce w Symfony i Spring Boot. Lepszym zastosowaniem dla tego szkieletu jest wystawianie punktów końcowych API, niż korzystanie z niego do budowy systemu w architekturze monolit MVC.

Przeprowadzone badania stanowią otwartą podstawę do dalszych prac oraz eksploracji innych możliwości badawczych. W przyszłości istnieje potencjał rozszerzenia analiz o bardziej zaawansowane aplikacje, charakteryzujące się zwiększoną liczbą punktów końcowych. Dodatkowo można sprawdzić mechanizmy zabezpieczeń w innych szkieletach oraz technologiach tj. np. Django czy ASP.NET Core, co pozwoli na przeprowadzenie bardziej szczegółowej i precyzyjnej analizy statystycznej uzyskanych wyników.

Literatura

[1] OWASP Top 10:2021, https://owasp.org/Top10/, [10.04.2025].

[2] Z. Čović, Threats and Vulnerabilities in Web Applications and How to Avoid Them, Critical Infrastructure Protection in the Light of the Armed Conflicts (2024) 93-103, https://doi.org/10.1007/978-3-031-47990-8_9.

[3] K. Nirmal, B. Janet, R. Kumar, Web Application Vulnerabilities - The Hacker's Treasure, In International Conference on Inventive Research in Computing Applications (2018) 56-62, http://dx.doi.org/10.1109/ICIRCA.2018.8597221.

[4] J. Adamu, R. Hamzah, M. M. Rosli, Security issues and framework of electronic medical record: A review, Bulletin of Electrical Engineering and Informatics (BEEI) 9(2) (2020) 565-572, https://doi.org/10.11591/eei.v9i2.2064.

[5] X. Mao, Comparison between Symfony, ASP.NET MVC, And Node.js Express for Web Development, Master thesis, North Dakota State University, Fargo, 2018.

[6] Z. Yin, SUJ Lee, Security Analysis of Web Open-Source Projects Based on Java and PHP, Electronics 12(12) (2023) 2618, https://doi.org/10.3390/electronics12122618.

[7] S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52(3/4) (1965) 591–611, https://doi.org/10.2307/2333709.

[8] O. J. Dunn, Multiple comparisons using rank sums, Technometrics 6(3) (1964) 241–252. https://doi.org/10.1080/00401706.1964.10490181.

[9] Dokumentacja języka programistycznego PHP, https://www.php.net/docs.php, [15.05.2025].

[10] Dokumentacja szkieletu programistycznego Symfony, https://symfony.com/doc/current/index.html, [15.05.2025].

[11] Dokumentacja szkieletu programistycznego Spring Boot, https://docs.spring.io/spring-boot/index.html, [15.05.2025].

[12] Dokumentacja szkieletu programistycznego Express, https://devdocs.io/express/, [15.05.2025].