ZAPORA SIECIOWA DO OCHRONY APLIKACJI INTERNETOWYCH Z WYKORZYSTANIEM UCZENIA MASZYNOWEGO

european student magazine
Numer 1 / 2025

Jakub **Zapała**[1], Aleksandra **Sikora**[2*]

[1] Politechnika Świętokrzyska, Wydział Elektrotechniki, Informatyki i Automatyki, al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Polska, s092840@student.tu.kielce.pl
[2] Politechnika Świętokrzyska, Wydział Elektrotechniki, Informatyki i Automatyki, al. Tysiąclecia Państwa Polskiego 7, 25-314 Kielce, Polska, asikora@tu.kielce.pl
[*] Corresponding author

Streszczenie – W artykule przedstawiono projekt i realizację zapory sieciowej typu Web Application Firewall (WAF), wykorzystującej techniki uczenia maszynowego do ochrony aplikacji internetowych. System został napisany w języku Python, z wykorzystaniem technologii FastAPI i Uvicorn do budowy logiki biznesowej oraz biblioteki PyQt5 do opracowania graficznego interfejsu użytkownika. W ramach pracy zaprojektowano klasyfikatory analizujące żądania HTTP, umożliwiające wykrywanie i blokowanie popularnych ataków, takich jak wstrzyknięcia SQL czy ataki typu XSS. System został przetestowany przy użyciu różnych algorytmów klasyfikacji, a skuteczność wykrywania ataków potwierdzono za pomocą dedykowanych testów.

Słowa kluczowe – zapora, uczenie maszynowe, bezpieczeństwo aplikacji internetowych, podatności

WSTĘP

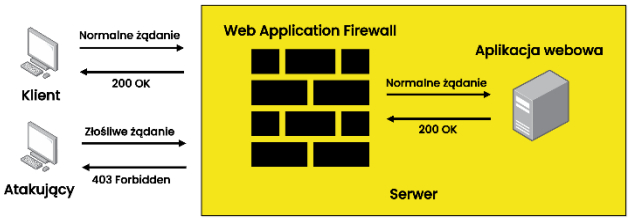
W ostatnich latach odnotowano znaczny wzrost liczby ataków skierowanych na warstwę aplikacji. Według danych Open Web Application Security Project [2], do najpowszechniejszych zagrożeń tego typu należą wstrzyknięcia SQL (ang. SQL Injection) oraz XSS (ang. Cross-Site Scripting).

Aby zmniejszyć ryzyko takich ataków, stosowane są różne metody ochrony, takie jak narzędzia do sanitacji danych oraz polityki bezpieczeństwa treści [2]. Jednym z podejść, na którym koncentruje się niniejszy artykuł, jest wykorzystanie zapory z funkcją wykrywania włamań do ochrony przed atakami na warstwę aplikacji. Tradycyjne systemy wykrywania włamań (ang. Intrusion Detection System, IDS) zazwyczaj opierają się na predefiniowanych regułach, co umożliwia atakującym ich ominięcie. Aby uczynić wykrywanie włamań bardziej dynamicznym i zdolnym do identyfikowania ataków zero-day [1], w pracy zastosowano techniki uczenia maszynowego.

Kluczowym aspektem w realizacji systemu jest opracowanie i wybór cech do wykrywania różnych typów ataków. Praca koncentruje się na określeniu, które cechy są najbardziej efektywne, a które mniej użyteczne przy tworzeniu modeli klasyfikacyjnych, służących do wykrywania zagrożeń.

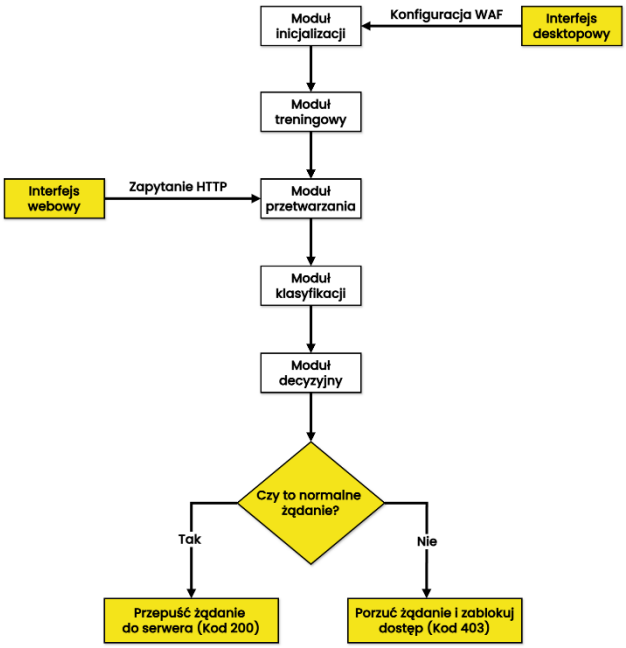
I. ARCHITEKTURA SYSTEMU

Architektura systemu nawiązuje do modelu WAF działającego jako warstwa pośrednicząca między klientem a serwerem. Rozwiązanie uruchamia się w formie procesu na serwerze, gdzie przechwytuje ruch HTTP przychodzący do chronionej aplikacji. Następnie przetwarza pozyskane dane, wyznacza wektor cech, dokonuje klasyfikacji oraz podejmuje decyzję o ewentualnym zablokowaniu (Rys. 1).



Rys. 1. Koncepcja architektury systemu

Koncepcja WAF dzieli się na pięć głównych modułów (Rys. 2). W module inicjalizacji pobierane są parametry wprowadzone przez użytkownika, takie jak adres IP i numer portu, a także typ algorytmu uczenia maszynowego. Po uruchomieniu zapory startuje moduł treningowy, który na podstawie zbioru HTTPParams [3] ładuje i trenuje wybrany klasyfikator, zapisując gotowy model i skaler w postaci plików. W momencie przyjścia żądania HTTP, moduł przetwarzania danych rozbija je na zdefiniowane cechy opisane w rozdziale II, a otrzymany w ten sposób wektor cech trafia do modułu klasyfikacji, gdzie uprzednio wytrenowany model ocenia, czy żądanie jest bezpieczne, czy też złośliwe. Ostatni etap stanowi moduł decyzyjny, który na podstawie wyniku klasyfikacji albo przepuszcza ruch do właściwego serwera, albo natychmiast go blokuje z kodem statusu 403 Forbidden (Permission Denied), eliminując potencjalne próby ataków jeszcze na poziomie warstwy aplikacyjnej.



Rys. 2. Schemat blokowy głównych modułów systemu

Interfejs desktopowy umożliwia wprowadzenie parametrów konfiguracyjnych oraz zarządzanie procesem trenowania i wyborem algorytmu uczenia maszynowego. Z kolei interfejs webowy pozwala użytkownikom na przeprowadzanie testów działania zapory, w tym przesyłanie przykładowych żądań i otrzymywanie informacji o ich klasyfikacji.

II. Techniki Przetwarzania Danych

W opracowanym narzędziu zdefiniowano zestaw metod przetwarzających treść żądania HTTP w celu uzyskania wektora cech, który został dobrany na podstawie analizy metryk wykorzystanych w podobnych rozwiązaniach [8, 9]. Każda przychodząca zawartość (ang. payload) jest analizowana znak po znaku, a następnie wyliczane są metryki, które pozwalają algorytmowi uczenia maszynowego odróżnić żądania normalne od złośliwych.

Ogólny wektor cech w przedstawia się następująco:

$$w = [l, s, d, a, k, \mu_b, \sigma_b, D] \quad (1)$$

gdzie:
l – długość zawartości,
s – proporcja znaków specjalnych,
d – proporcja cyfr,
a – proporcja znaków alfanumerycznych,
k – proporcja słów kluczowych,
μ_b – średnia wartość bajtów,
σ_b – odchylenie standardowe bajtów,
D – liczba unikalnych bajtów.

Do nauczenia modeli uczenia maszynowego wykorzystano zbiór danych HTTPParams, zawierający przykłady żądań sklasyfikowanych jako normalne lub złośliwe. Zbiór ten obejmuje 19 304 normalnych żądań oraz 11 763 złośliwych żądań, w tym 10 852 wstrzyknięć SQL, 532 ataków typu XSS, 290 ataków typu Path Traversal oraz 89 wstrzyknięć poleceń.

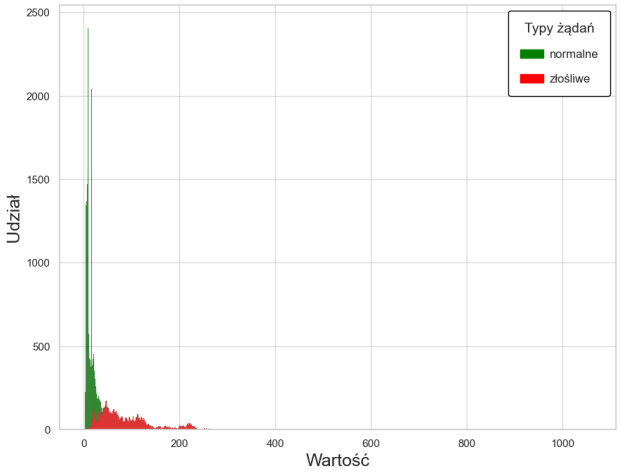
Długość Zawartości

Opisuje liczbę znaków w żądaniu:

$$l = \sum_{i=0}^{n} c_i \quad (2)$$

gdzie:
l – długość zawartości,
c_i – kolejny znak w zawartości,
n – indeks ostatniego znaku.

Na podstawie wykonanego histogramu (Rys. 3) stwierdzono, że wartość tej cechy w żądaniach złośliwych jest zazwyczaj wyższa niż w żądaniach normalnych.



Rys. 3. Histogram długości zawartości w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonej długości, wartość – reprezentuje długość żądań wyrażoną w liczbie znaków.

Średnia cechy wynosi 41, a odchylenie standardowe 56. Minimalna długość zawartości wynosiła tylko jeden znak, a maksymalna miała 1058 znaków.

Proporcja Znaków Specjalnych

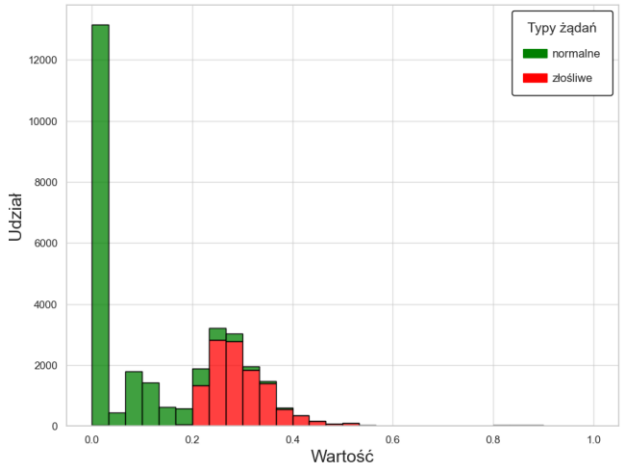
Opisuje stosunek znaków specjalnych (niealfanumerycznych) do długości żądania:

$$s = \sum_{i=0}^{n} \frac{(c_i \mid c_i \in O)}{l} \quad (3)$$

gdzie:
s – proporcja znaków specjalnych,
c_i – kolejny znak w zawartości,
O – zbiór znaków, które nie są literą ani cyfrą,
l – długość zawartości,
n – indeks ostatniego znaku.

Na podstawie wykonanego histogramu (Rys. 4) stwierdzono, że złośliwe żądania zwykle zawierają mniej

znaków alfanumerycznych, a więcej znaków specjalnych. W rezultacie ta cecha przyjmuje wyższe wartości w przypadku żądań złośliwych.



Rys. 4. Histogram znaków specjalnych w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonym poziomie współczynnika znaków specjalnych, wartość – reprezentuje stosunek liczby znaków specjalnych do całkowitej liczby znaków w żądaniu.

W większości przypadków liczba znaków specjalnych w żądaniu nie przekracza wartości 0,14, natomiast odnotowano pojedyncze sytuacje ze współczynnikiem równym 1, co oznacza całkowite zdominowanie przez znaki specjalne.

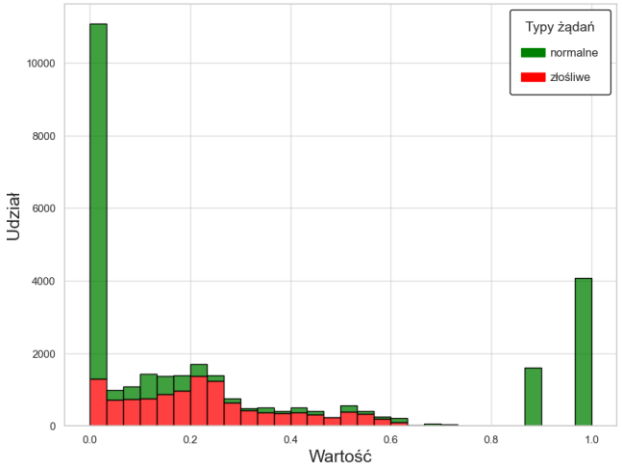
PROPORCJA CYFR

Opisuje stosunek cyfr do długości żądania:

$$d = \sum_{i=0}^{n} \frac{(c_i \mid c_i \in U)}{l} \tag{4}$$

gdzie:
d – proporcja cyfr,
c_i – kolejny znak w zawartości,
U – zbiór znaków, zawierający same cyfry,
l – długość zawartości,
n – indeks ostatniego znaku.

Na podstawie wykonanego histogramu (Rys. 5) stwierdzono, że wartość tej cechy jest przeważnie wyższa w żądaniach złośliwych niż w żądaniach normalnych. Wynika to z faktu, że niektóre ataki, takie jak wstrzyknięcia SQL, cechują się nadmiernym użyciem cyfr.



Rys. 5. Histogram cyfr w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonym poziomie współczynnika cyfr, wartość – reprezentuje stosunek liczby cyfr do całkowitej liczby znaków w żądaniu.

Przeciętna proporcja wynosi niespełna 0,29, jednak duże odchylenie standardowe wynoszące 0,35 świadczy o znacznym zróżnicowaniu zapytań.

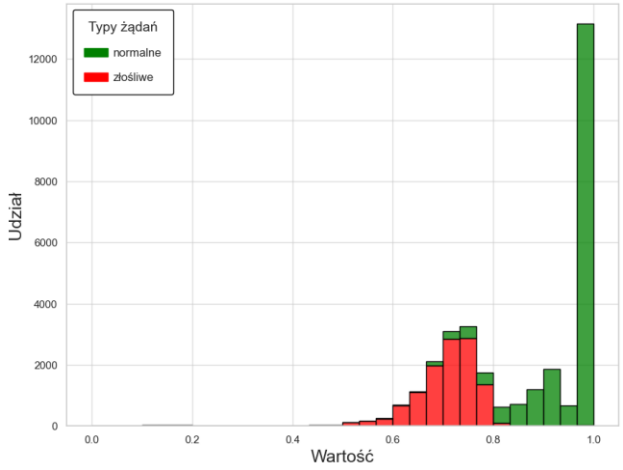
PROPORCJA ZNAKÓW ALFANUMERYCZNYCH

Opisuje stosunek znaków alfanumerycznych do długości żądania:

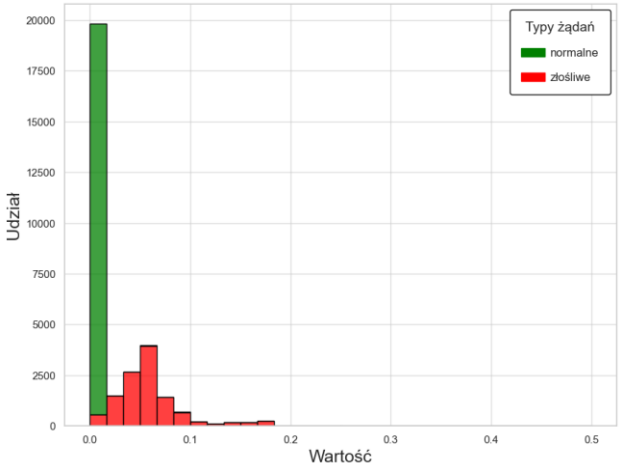
$$a = \sum_{i=0}^{n} \frac{(c_i \mid c_i \in A)}{l} \tag{5}$$

gdzie:
a – proporcja znaków alfanumerycznych,
c_i – kolejny znak w zawartości,
A – zbiór znaków, zawierający litery i cyfry,
l – długość zawartości,
n – indeks ostatniego znaku.

Na podstawie wykonanego histogramu (Rys. 6) zauważono, że żądania normalne przeważnie zawierają więcej znaków alfanumerycznych niż znaków specjalnych. W rezultacie ta cecha osiąga wyższe wartości dla żądań normalnych.



Rys. 6. Histogram znaków alfanumerycznych w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonym poziomie współczynnika znaków alfanumerycznych, wartość – reprezentuje stosunek liczby znaków alfanumerycznych do całkowitej liczby znaków w żądaniu.



Rys. 7. Histogram słów kluczowych w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonym poziomie współczynnika słów kluczowych, wartość – reprezentuje stosunek liczby słów kluczowych do całkowitej liczby słów w żądaniu.

Średnia cechy wynosi 0,86, co wskazuje, że w wielu przypadkach większość znaków to symbole alfanumeryczne. Mediana na poziomie 0,9 sugeruje, że ponad połowa żądań zawiera co najmniej 90% znaków alfanumerycznych.

PROPORCJA SŁÓW KLUCZOWYCH

Określa udział słów atakujących w całkowitej długości żądania:

$$k = \sum_{i=0}^{n} \frac{(w_i \mid w_i \in X)}{l} \tag{6}$$

gdzie:
k – proporcja słów kluczowych,
w_i – kolejne słowo w zawartości,
X – zbiór najpopularniejszych słów atakujących,
l – długość zawartości,
n – liczba słów w zawartości.

Wartość tej cechy inicjalizowana jest zerem i wzrasta dla każdego znalezionego słowa atakującego w treści zawartości (Rys. 7).

W większości przypadków złośliwe żądania osiągają wyraźnie wyższe wartości w porównaniu do żądań normalnych, dla których ta cecha przyjmuje wartość 0.

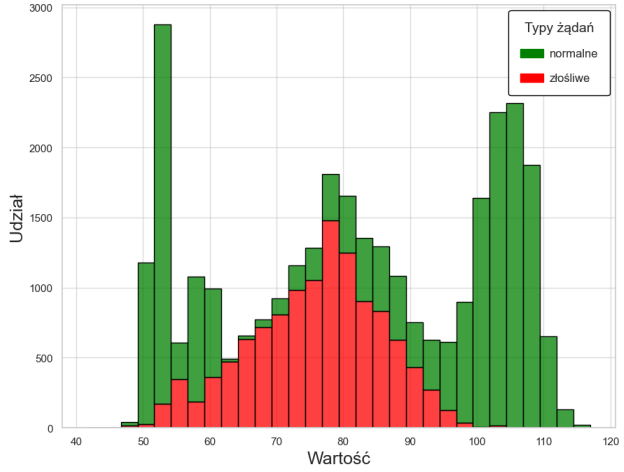
ŚREDNIA WARTOŚĆ BAJTÓW

Opisuje średnią wartość liczb porządkowych znaków (ASCII) przekonwertowanych na reprezentację bajtową, które wchodzą w skład żądania:

$$\mu_b = \sum_{i=0}^{n} \frac{b_i}{n} \tag{7}$$

gdzie:
μ_b – średnia wartość bajtów,
b_i – kolejna wartość bajtu w zawartości,
n – liczba wszystkich bajtów w zawartości.

Na podstawie wykonanego histogramu (Rys. 8) zauważono, że w złośliwych żądaniach średnia jest przesunięta w inną stronę niż w żądaniach normalnych, w związku z tym, że w złośliwych żądaniach z reguły pojawiają się nietypowe symbole.



Rys. 8. Histogram średniej wartości bajtów w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonym poziomie średniej wartości bajtów, wartość – reprezentuje średnią wartość bajtów wyrażoną w kodach ASCII.

Większość złośliwych żądań ma wartości oscylujące wokół 78–82. Widoczny jest szczyt wokół wartości 50 i 105, co odpowiada normalnym żądaniom.

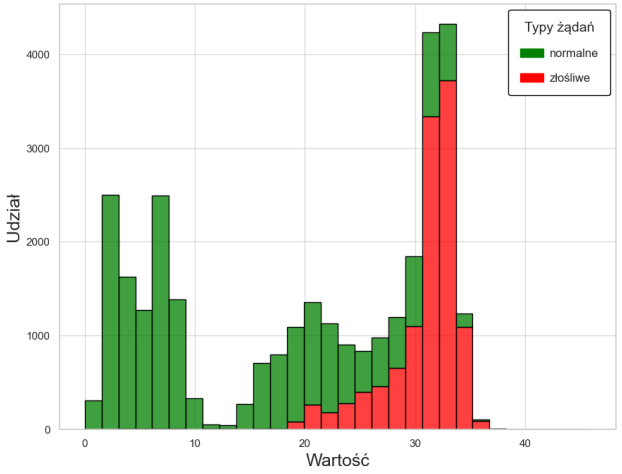
ODCHYLENIE STANDARDOWE BAJTÓW

Opisuje rozproszenie wartości bajtowych w żądaniu względem ich średniej:

$$\sigma_b = \sum_{i=0}^{n} \sqrt{\frac{(b_i - \mu_b)^2}{n}} \tag{8}$$

gdzie:
σ_b – odchylenie standardowe bajtów,
μ_b – średnia wartość bajtów,
b_i – kolejna wartość bajtu w zawartości,
n – liczba wszystkich bajtów w zawartości.

Na podstawie wykonanego histogramu (Rys. 9) stwierdzono, że w normalnych żądaniach składających się głównie z wąskiej grupy znaków odchylenie jest relatywnie niewielkie, natomiast w przypadku złośliwych żądań składających się zazwyczaj z wielu różnych symboli, w tym znaków nietypowych, wartość tej cechy jest większa.



Rys. 9. Histogram odchylenia standardowego bajtów w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonym poziomie odchylenia standardowego bajtów, wartość – reprezentuje odchylenie standardowe bajtów wyrażone w kodach ASCII.

W wielu przypadkach występują umiarkowane wartości rzędu 7–31, lecz odnotowano również zawartości o odchyleniu powyżej 40, co wiąże się z bardzo szerokim zakresem wykorzystywanych bajtów.

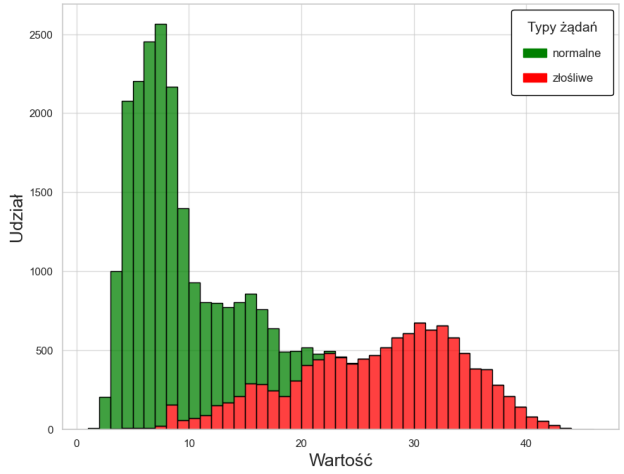
LICZBA UNIKALNYCH BAJTÓW

Określa, ile odmiennych wartości bajtowych zostało użytych w żądaniu:

$$D = |\{b_1, b_2, \dots, b_i\}| \tag{9}$$

gdzie:
D – liczba unikalnych bajtów,
b_i – kolejna wartość bajtu w zawartości.

Na podstawie wykonanego histogramu (Rys. 10) zauważono, że dla złośliwych żądań liczba unikalnych bajtów bywa znacznie większa niż w przypadku żądań normalnych.



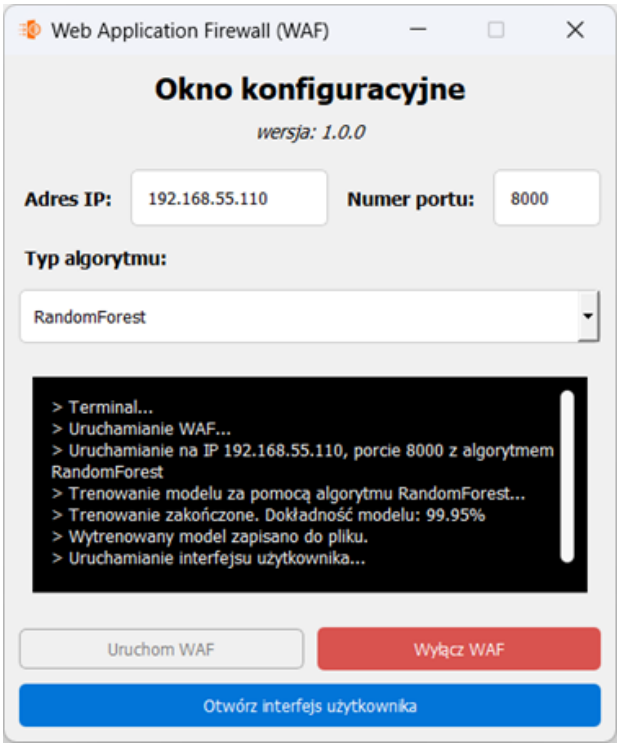
Rys. 10. Histogram liczby unikalnych bajtów w zbiorze danych HTTPParams, gdzie udział – oznacza liczbę wystąpień żądań o określonej liczbie unikalnych bajtów, wartość – reprezentuje liczbę różnych wartości bajtowych wykorzystanych w żądaniu.

W przeważającej części żądań wartości wahają się w okolicach 6–23, natomiast w ekstremalnych przypadkach zidentyfikowano aż 45 różnych bajtów. Taka różnorodność bywa znacznie wyższa w żądaniach atakujących niż w ruchu normalnym.

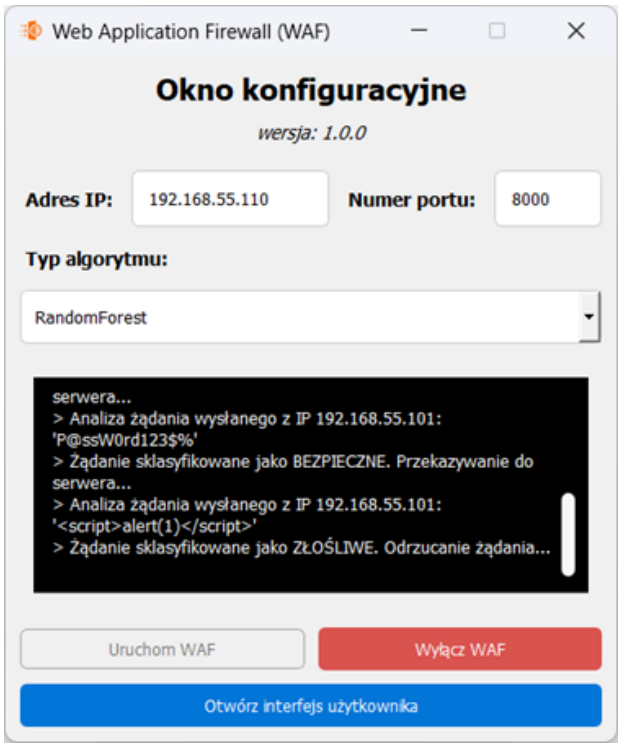
III. Interfejs Użytkownika

Interfejs użytkownika podzielono na dwa odrębne komponenty. Pierwszy to aplikacja desktopowa napisana w PyQt [4], umożliwiająca wprowadzenie parametrów konfiguracyjnych zapory, takich jak adres IP czy numer portu. Użytkownik może tam także wybrać typ algorytmu uczenia maszynowego i przeprowadzić jego trenowanie na przygotowanym zbiorze danych. Dostępny jest również przycisk otwierający panel webowy w domyślnej przeglądarce (Rys. 11). Zaimplementowano tutaj walidację wpisanego adresu IP oraz numeru portu w celu uniknięcia błędów związanych z uruchomieniem serwera.

Po poprawnej konfiguracji w części ekranu konfiguracyjnego wyświetlane są logi prezentujące informacje o ruchu przychodzącym do zapory (Rys. 12). Każde żądanie wygenerowane z interfejsu webowego jest rejestrowane wraz z jego treścią, adresem IP, z którego pochodzi, oraz wynikiem klasyfikacji.



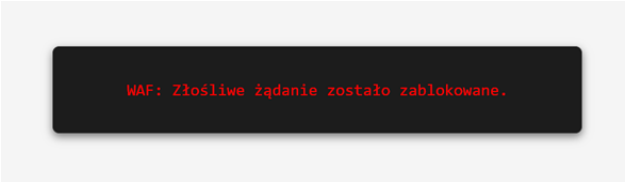
Rys. 11. Ekran konfiguracyjny aplikacji desktopowej



Rys. 12. Prezentowanie logów o ruchu przychodzącym na ekranie konfiguracyjnym

Drugi komponent to interfejs webowy zrealizowany w bibliotece React [5]. Umożliwia on przeprowadzenie testów przez wprowadzenie przykładowej zawartości (ang. payload) w formularzu. Żądanie kierowane jest do endpointu w logice biznesowej zapory, który dokonuje klasyfikacji i zwraca odpowiedni komunikat, gdy żądanie jest bezpieczne (Rys. 13) lub złośliwe (Rys. 14).

Rys. 13. Komunikat aplikacji webowej, gdy żądanie zostało uznane za bezpieczne



Rys. 14. Komunikat aplikacji webowej, gdy żądanie zostało uznane za złośliwe

IV. Testy i Analiza Wyników

W celu zweryfikowania skuteczności zaprojektowanego systemu, zrealizowano procedurę trenowania trzech różnych algorytmów uczenia maszynowego [10]: las losowy (ang. Random Forest), wektory nośne (ang. Support Vector Machines, SVM) oraz regresja logistyczna (ang. Logistic Regression), wykorzystując do tego zbiór HTTPParams. Dla lasu losowego zastosowano następujące parametry: 100 drzew (n_estimators=100), brak ograniczenia głębokości drzew (max_depth=None), minimalną liczbę próbek do podziału węzła równą 2 (min_samples_split=2), minimalną liczbę próbek w liściu równą 1 (min_samples_leaf=1) oraz dobór liczby cech do podziału zgodnie z metodą *sqrt* (max_features='sqrt'). Dla wektorów nośnych wybrano jądro gaussowskie (kernel='rbf'), parametr regularyzacji równy 1 (C=1.0) oraz skalowanie parametru gamma według liczby cech (gamma='scale'). W regresji logistycznej przyjęto ustawienia z maksymalną liczbą iteracji równą 100 (max_iter=100), parametrem regularyzacji 1 (C=1.0) oraz optymalizatorem *liblinear* (solver='liblinear').

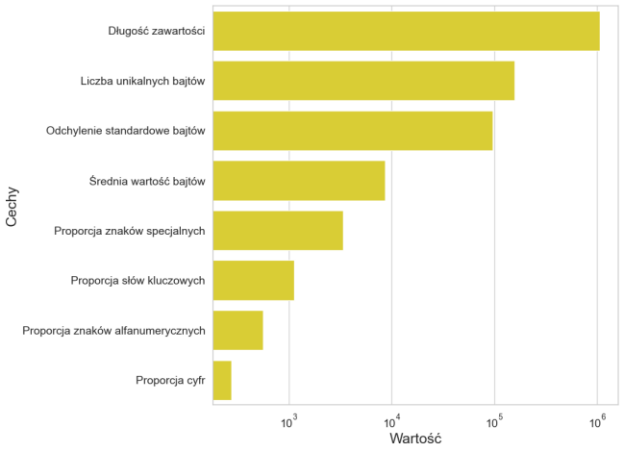
Metryki do oceny jakości klasyfikacji wyliczono na podstawie danych testowych, stanowiących 30% całego zbioru (Tabela 1), zgodnie ze standardowymi definicjami stosowanymi w literaturze z zakresu uczenia maszynowego [11], gdzie dokładność – oznacza ogólną skuteczność klasyfikatora, precyzja – wskazuje, jak dokładnie klasyfikator identyfikuje pozytywne przypadki, czułość – reprezentuje zdolność klasyfikatora do wykrywania pozytywnych przypadków, swoistość – reprezentuje zdolność klasyfikatora do identyfikowania negatywnych przypadków oraz F1 – stanowi średnią harmoniczną precyzji i czułości.

Tabela 1. Porównanie modeli uczenia maszynowego

Model	Dokładność	Precyzja	Czułość	Swoistość	F1
Las losowy	99,95%	99,93%	99,98%	99,88%	99,96%
Wektory nośne	99,78%	99,89%	99,77%	99,79%	99,83%
Regresja logistyczna	99,60%	99,74%	99,65%	99,52%	99,69%

Uzyskane wyniki wskazują, że przy wystarczająco rozbudowanym zbiorze danych i dobrze dobranych cechach, wszystkie trzy algorytmy mogą osiągać wyniki powyżej 99,5% dokładności. Niemniej jednak las losowy wykazał się najlepszą stabilnością i marginalnie wyższymi wartościami precyzji i czułości.

W celu określenia skuteczności poszczególnych cech w procesie klasyfikacji zastosowano metodę *SelectKBest()* z biblioteki scikit-learn [6]. Metoda ta umożliwia identyfikację zmiennych, które w największym stopniu przyczyniają się do rozróżniania próbek złośliwych od normalnych, poprzez przeprowadzenie oceny statystycznej cech za pomocą testu chi-kwadrat [7].



Rys. 15. Ranking cech według SelectKBest()

Najistotniejsze okazały się cechy powiązane z długością żądania oraz różnorodnością znaków, podczas gdy proporcja cyfr charakteryzowała się mniejszą przydatnością (Rys. 15). Połączenie wielu cech, takich jak proporcja znaków specjalnych i słów kluczowych, skutecznie wspomagało klasyfikację i pozwalało na ograniczenie zarówno liczby fałszywych alarmów, jak i nieuchwyconych ataków.

V. WNIOSKI

Opracowana zapora, bazująca na technikach uczenia maszynowego, skutecznie rozwiązuje problem ochrony warstwy aplikacyjnej przed najpopularniejszymi typami ataków, takimi jak wstrzyknięcia SQL, XSS, Path Traversal czy wstrzyknięcia poleceń. Zaprojektowany prototyp integruje logikę biznesową zrealizowaną w FastAPI i Uvicorn z interfejsem webowym React oraz aplikacją desktopową PyQt, co zapewnia elastyczność konfiguracji i wysoką skuteczność w blokowaniu złośliwego ruchu.

Zapora WAF umożliwia trenowanie wybranego modelu uczenia maszynowego na przygotowanym zbiorze danych, klasyfikowanie żądań HTTP, blokowanie dostępu w przypadku wykrycia podejrzanego ruchu oraz rejestrowanie wszystkich zdarzeń w postaci logów Interfejs webowy umożliwia szybkie testowanie działania zapory (Rys. 13 i 14), a interfejs desktopowy pozwala na konfigurowanie parametrów, takich jak port, adres IP czy typ algorytmu (Rys. 11 i 12).

Przeprowadzone testy potwierdziły wysoką efektywność przy wykrywaniu różnych form ataków, a wszystkie zastosowane algorytmy, takie jak las losowy, wektory nośne oraz regresja logistyczna, osiągnęły dokładność powyżej 99,5% (Tabela 1). Testy przeprowadzone przy użyciu zewnętrznych bibliotek wykazały, że zastosowanie szerokiego wachlarza cech znacząco poprawiło skuteczność klasyfikacji, redukując zarówno liczbę fałszywych alarmów, jak i nierozpoznanych ataków (Rys. 15).

FIREWALL FOR THE PROTECTION OF WEB APPLICATIONS USING MACHINE LEARNING

This article presents a Web Application Firewall (WAF) prototype that uses machine learning to protect modern web applications from common threats such as SQL Injection and Cross-Site Scripting. The system incorporates a Python-based backend running under FastAPI and a React-based user interface, along with a desktop application for configuration. The proposed design analyzes incoming HTTP requests by extracting multiple features, including request length and distinctive character ratios, to detect malicious payloads. Various classification algorithms, such as Random Forest, Support Vector Machine, and Logistic Regression, were compared on a dataset of labeled requests, achieving accuracy rates above 99.5 percent. Thanks to its modular structure, the WAF prototype can be extended with new machine learning models or additional protection features and integrated into diverse environments.

Key words: firewall, machine learning, web application security, vulnerabilities

BIBLIOGRAFIA

[1] Tuz M. (2023), Wpływ cyberzagrożeń na funkcjonowanie organizacji. Przegląd Policyjny, Vol 151, Issue 3, pp 29, doi: 10.5604/01.3001.0053.9773

[2] Sucuri, OWASP Top Security Risks & Vulnerabilities, https://sucuri.net/guides/owasp_top_10_2021_edition/ [data publikacji: 2021, data dostępu: 2024.12.04]

[3] Morzeux, HttpParamsDataset, https://github.com/Morzeux/HttpParamsDataset [data ostatniej aktualizacji: 2016, data dostępu: 2024.11.30]

[4] PyQt5, Dokumentacja biblioteki PyQt5, https://pypi.org/project/PyQt5/ [data dostępu: 2024.12.20]

[5] React, Dokumentacja frameworka React, https://react.dev/ [data dostępu: 2024.12.20]

[6] Scikit-learn, Dokumentacja biblioteki scikit-learn, https://scikit-learn.org/ [data dostępu: 2024.12.20]

[7] McHugh ML. (2013), The Chi-square test of independence, Biochem Med, pp 145, doi: 10.11613/BM.2013.018

[8] Shaheed A., Kurdy M. H. D. B. (2022), Web Application Firewall Using Machine Learning and Features Engineering. Security Comm. Networks, Vol 2022, Issue 1, pp 7, doi: 10.1155/2022/5280158

[9] Torrano-Gimenez C., Nguyen H. T., Alvarez G., Franke K. (2012), Combining expert knowledge with automatic feature extraction for reliable web attack detection. Security Comm. Networks, Vol 8, Issue 16, pp 2756, doi: 10.1002/sec.603

[10] Siddik AB., Badal FR., Islam A. (2024), Comparative performance of machine learning algorithms for early genetic disorder and subclass classification, pp 6-7, doi: 10.48550/arXiv.2412.0218

[11] Bidve, P., Mishra, S., Annapurna J. (2023), Enhancing Ayurvedic Diagnosis using Multinomial Naive Bayes and K-modes Clustering: An Investigation into Prakriti Types and Dosha Overlapping, pp 4, doi: 10.48550/arXiv.2310.02920