Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Rozprawa Doktorska

Orkiestracja narzędzi bezpieczeństwa w sieci operatora telekomunikacyjnego z wykorzystaniem technik uczenia maszynowego oraz metod przetwarzania języka naturalnego

Grzegorz Siewruk
Numer albumu 6461

promotor
dr hab.inż. Wojciech Mazurczyk, prof. PW
promotor pomocniczy
dr inż. Adrian Marzecki

WARSZAWA 2021

Streszczenie

Orkiestracja narzędzi bezpieczeństwa w sieci operatora telekomunikacyjnego z wykorzystaniem technik uczenia maszynowego oraz metod przetwarzania języka naturalnego

Coraz więcej zespołów wytwarzających oprogramowanie korzysta obecnie z rozwiązań pozwalających na automatyzację większości czynności, które jeszcze kilka lat temu były wykonywane manualnie. Tego typu zmiany pozwalają na zwiększenie elastyczności procesów wytwarzania oprogramowania oraz umożliwiają szybsze wdrażanie zmian w oprogramowaniu i ich udostępnianiu użytkownikowi końcowemu. Niestety prowadzi to również do pewnych niedogodności. Największą z nich jest problem z zapewnieniem bezpieczeństwa tak zbudowanemu procesowi, w którym kod źródłowy aplikacji może się zmieniać nawet kilkanaście razy dziennie. Do dyspozycji zespołów bezpieczeństwa jest różnego rodzaju oprogramowanie, pozwalające na wykonywanie skanów podatności (w tym modelu realizacją manualnych testów bezpieczeństwa nie jest możliwa z uwagi na wolumen zmian).

Niestety automatyczne wykonywanie skanów podatności wiąże się ze zgłaszaniem (przez narzędzia) bardzo dużej liczby naruszeń, które muszą być odpowiednio przeanalizowane. Zarządzanie podatnościami bezpieczeństwa raportowanymi przez różnego rodzaju oprogramowanie skanujące jest skomplikowanym i czasochłonnym zajęciem. Konkretne błędy bezpieczeństwa mogą mieć różny wpływ na system teleinformatyczny w zależności od kontekstu, w jakim on działa. Co więcej, liczba błędów zgłaszanych przez konkretne rozwiązania może być bardzo duża (od kilkuset do kilku tysięcy), co zdecydowanie utrudnia analizę oraz określanie odpowiednich priorytetów dotyczących tego, które błędy należy usunąć w pierwszej kolejności.

Aby rozwiązać przedstawione powyżej problemy, w niniejszej rozprawie przedstawiono koncepcję rozwiązania *Mixeway*, które zostało wdrożone w infrastrukturze rzeczywistego operatora telekomunikacyjnego. Wspomniany system jest odpowiedzialny za orkiestrację pracy narzędzi bezpieczeństwa oraz umożliwia klasyfikację wykrytych podatności. Wykorzystany w *Mixeway* klasyfikator został przygotowany w procesie nauczania opartego o rzeczywiste dane zbierane przez 12 miesięcy z produkcyjnie działających aplikacji. Zebrane zagrożenia klasyfikowane są do jednej z dwóch kategorii: konieczne do poprawy w badanym kontekście (ang. Confirmed and Relevant Vulnerability - CRV) oraz nieistotne w badanym kontekście (ang. Detected but Not Relevant Vulnerability - DRNV). Wyniki uzyskane przez opracowany system *Mixeway* pozwalają potwierdzić skuteczność przedstawionego rozwiązania w kontekście automatycznej weryfikacji zagrożeń w procesie dostarczania oprogramowania.

Słowa kluczowe: bezpieczeństwo IT, CICD, devsecops, uczenie maszynowe

Abstract

Security tools orchestration in mobile network operator environment using neural networks and natural language processing techniques

Increasingly, development teams are using software that automates operations which a few years back were performed manually. Thanks to that, the agility of the team is increased, which enables the possibility to decrease the time needed to run the software on the production environment available for the end user. Unfortunately, it also leads to some inconveniences. The main issue is the problem with providing security to such a process, in which the source code of an application may change even a dozen times a day. The security teams have at their disposal various software that allows to perform vulnerability scans (in this model, manual security testing is not possible due to the volume of changes).

Unfortunately, performing vulnerability scans in an automated manner involves reporting (by tools) a very large number of potential breaches that must be analyzed properly. Managing security vulnerabilities reported by various types of scanning software is a complex and time-consuming activity. Especially in organizations that produce a lot of software. Specific security vulnerabilities can have a different impact on an IT system depending on the context in which it operates. What is more, the number of vulnerabilities reported by specific solutions can be very high (from a few hundred to a few thousand), which makes it difficult to analyze and prioritize which vulnerabilities should be fixed first.

To solve the problems mentioned above, in this dissertation we introduce the concept of Mixeway solution, which has been implemented in the infrastructure of a real-world telecommunications operator. The mentioned system orchestrates the work of security tools and, thanks to the implemented model using machine learning algorithms along with natural language processing techniques, enables the classification of detected security vulnerabilities. The classifier was prepared in a learning process based on data collected for 12 months from production applications. The vulnerabilities are classified into two categories: Confirmend and Relevant Vulnerability (CRV) and Detected but Not Relevant Vulnerability (DRNV). The results obtained by the developed classifier allow to confirm the effectiveness of the presented solution in the context of automatic verification of threats in the software delivery process.

Keywords: IT security, cicd, devsecops, machine learning

Spis treści

Streszczenie **3**

Abstract **5**

Publikacje, prezentacje i nagrody związane z tematyką rozprawy **10**

Lista skrótów **12**

1. Wprowadzenie **21**
1.1. Wstęp . . . 21
1.2. Cele i teza rozprawy . . . 24
1.3. Układ pracy . . . 24

2. Przegląd istniejących rozwiązań **26**
2.1. Wykrywanie ataków sieciowych w oparciu o mechanizmy uczenia maszynowego . . . 26
2.1.1. Systemy wykrywania włamań . . . 26
2.1.2. Ochrona urządzeń końcowych . . . 27
2.2. Orkiestracja narzędzi bezpieczeństwa . . . 28
2.3. Wykrywanie podatności w oprogramowaniu w oparciu o algorytmy uczenia maszynowego . . . 29
2.4. Klasyfikacja wykrytych podatności z wykorzystaniem algorytmów uczenia maszynowego . . . 30
2.5. Rozwiązania komercyjne . . . 31
2.6. Podsumowanie . . . 31

3. Metodyki prowadzenia projektów IT **34**
3.1. Cykl życia projektu . . . 34
3.2. Model kaskadowy . . . 36
3.3. Model zwinny . . . 36
3.4. Miejsce bezpieczeństwa w procesie dostarczania oprogramowania . . . 37

4. Klasyfikacja z wykorzystaniem algorytmów uczenia maszynowego oraz techniki przetwarzania języka naturalnego **42**
4.1. Las losowy . . . 43
4.2. Sieci neuronowe . . . 44
4.3. SVM . . . 47
4.4. Przetwarzanie języka naturalnego . . . 49

5. Mixeway: architektura i sposób wdrożenia **51**

5.1. Architektura 52
5.1.1. Wykrywanie zasobów 54
5.1.2. Zarządzanie skanami bezpieczeństwa 56
5.1.3. Korelacja wyników 58
5.2. Etapy wdrożenia rozwiązania 60
5.2.1. Etap 1: Opracowanie procedury zarządzania podatnościami 61
5.2.2. Etap 2: Wdrożenie modułu detekcji zasobów 62
5.2.3. Etap 3: Wdrożenie modułu zarządzania testami bezpieczeństwa . . 62
5.2.4. Etap 4: Opracowanie i wdrożenie modułu korelacji wyników 63
5.2.5. Etap 5: Wdrożenie bramki bezpieczeństwa w łańcuchu dostarczania oprogramowania 64
5.3. Przedstawienie graficznego interfejsu użytkownika 65
5.4. Stan przed wdrożeniem rozwiązania w środowisku operatora telekomunikacyjnego 67
5.5. Stan po wdrożeniu rozwiązania w środowisku operatora telekomunikacyjnego 68

6. Metodyka badawcza **70**
6.1. Sposób przeprowadzenia badań eksperymentalnych 70
6.2. Opis danych 71
6.3. Przetwarzanie wstępne 74
6.4. Opis hiperparametrów badanych klasyfikatorów 77

7. Wyniki badań eksperymentalnych **81**
7.1. Badanie skuteczności modeli zbudowanych w oparciu o przedstawione algorytmy 82
7.2. Wyniki uzyskane dzięki wdrożeniu systemu Mixeway w infrastrukturze operatora telekomunikacyjnego 89

8. Podsumowanie **93**
8.1. Klasyfikacja podatności z wykorzystaniem algorytmów uczenia maszynowego 94
8.2. Wykorzystanie opracowanego modelu w realizacji zadań operacyjnych . . 95
8.3. Kierunki dalszych prac 95

Bibliografia **97**

Spis załączników **106**

Publikacje, prezentacje i nagrody związane z tematyką rozprawy

W ramach realizacji założeń projektu i pracy naukowej autora opublikowano dwa artykuły w języku angielskim, zaprezentowano wyniki na trzech konferencjach. Ponadto, opracowany w ramach rozprawy system *Mixeway* został uhonorowany dwoma nagrodami.

Ponadto, system, który został wdrożony w infrastrukturze Orange Polska, udostępniono na zasadzie licencji GPL-3.0 za pośrednictwem serwisu GitHub skąd pobrano go oraz zainstalowano ponad 15 000 razy (stan na 25.08.2021). W internetowej wersji demonstracyjnej zarejestrowano 112 użytkowników, którzy w badanych przez siebie aplikacjach wykryli 4 700 podatności bezpieczeństwa (stan na 25.07.2021).

Publikacje

- Siewruk G, Mazurczyk W. Context-Aware Software Vulnerability Classification using Machine Learning. IEEE Access. 2021 Apr 23.
- Siewruk G, Mazurczyk W, Karpiński A. Security Assurance in DevOps Methodologies and Related Environments. International Journal of Electronics and Telecommunications. 2019 May 19;65(2):211-6.

Konferencje

- Siewruk G. O tym jak (nie)łatwo dodać Sec do Dev(Sec)Ops w świecie telco. PLNOG19. Kraków 2019.
- Siewruk G. O tym jak mało jest Sec w Dev(Sec)Ops podczas testów aplikacji. WTH19. Warszawa 2019.
- Siewruk G. Klasyfikacja podatności jako weryfikacja w procesach automatycznego dostarczania oprogramowania. Advanced Threat Summit. Warszawa 2020.

Wspomnienia w prasie

- Orange Funds New CI/CD Security Tool, HardenStance Briefing, 2020.

Nagrody

- Nagroda szefa funkcji Sieć i Technologie 2019 (Załącznik 2) - przyznawana raz w roku za bardzo dobre wyniki lub wyjątkowe postawy w projektach o największym znaczeniu dla realizacji rocznych celów danej funkcji. Przyznawana przez Członka Zarządu ds.Sieci i Technologii Orange Polska.
- Security & Privacy Awards 2021 (Załącznik 3) - Coroczny konkurs nagradzający najlepsze rozwiązania, wpływające na bezpieczeństwo w Grupie Orange. Nagroda przyznawana jest przez jury, na którego czele stoi dyrektor wykonawczy ds. Strategii i Cyberbezpieczeństwa Grupy Orange.

Lista skrótów

Skrót	j. angielski	j. polski
API	Application Programming Interface	Interfejs programistyczny aplikacji
AWS	Amazon Web Services	-
BV	Bohr Vulnerability	Prosta podatność do usunięcia
C&C	Command and Control	Server sterujący złośliwym oprogramowaniem
CD	Continous Deployment	Proces ciągłego wdrożenia
CI	Continous Integration	Proces ciągłej integracji
CICD	Continous Integration Continous Deployment	Proces ciągłej integracji i ciągłego wdrożenia
CNN	Convolutional Neural Network	Konwolucyjne sieci neuronowe
CRV	Confirmed and Relevant Vulnerability	Potwierdzona i istotna podatność
CSV	Comma Separated Value	Plik z danymi separowane przecinkiem
CVE	Common Vulnerability & Exposures	Baza podatności
CVSS	Common Vulnerability Scoring System	Wspólny system oceny krytyczności podatności
DAST	Dynamic Application Security Testing	Testy dynamiczne aplikacji
DNRV	Detected but Not Relevant Vulnerability	Wykryta, ale nieistotna podatność
FN	False Negative	Wartość fałszywie negatywna
FP	False Positive	Wartość fałszywie pozytywna
GCP	Google Cloud Platform	-
HTTP	Hypertext Transfer Protocol	Protokół przesyłania dokumentów hipertekstowych
IAST	Interactive Application Security Testing	Testy interaktywne aplikacji
IDS	Intrusion Detection System	System wykrywania włamań

IP	Internet Protocol	-
LB	Load Balancing	System równoważenia obciążeń
LB	Naive Bayes	Naiwny klasyfikator bayesowski
LR	Logistic Regression	Regresja logiczna
LSTM	Long Short Term Memory	Pamięć z komórkami krótko i długo terminowymi
ML	Machine Learning	Uczenie Maszynowe
MV	Mandel Vulnerability	Skomplikowana podatność do usunięcia
NLP	Natural Language Processing	Przetwarzanie Języka Naturalnego
NN	Neural Network	Sieci neuronowe
NVD	National Vulnerability Database	Międzynarodowa baza podatności bezpieczeństwa
PCI DSS	Payment Card Industry Data Security Standard	Norma bezpieczeństwa wydana przez Payment Card Industry Security Standards Council
PM	Project Manager	Kierownik Projektu
POS	Part of Speech	Wyodrębnienie części mowy
RAST	Runtime Application Security Testing	Testy bezpieczeństwa podczas działania aplikacji
RF	Random Forrest	Las losowy
RNN	Recurrent Neural Network	Rekurencyjne sieci neuronowe
SAST	Static Application Security Testing	Testy statyczne kodu źródłowego
SDLC	Software Development Life Cycle	Cykl życia projektu IT
SOX	Overview of Sarbanes Oxley	Ustawa Sarbanesa-Oxleya
SSH	Secure Shell	Bezpieczne połączenie terminalowe
SVM	Support Vector Machine	Maszyny wektorów nośnych

TN	True Negative	Wartość prawdziwie negatywna
TP	True Positive	Wartość prawdziwie pozytywna
XSS	Cross-site Scripting	-

Spis tabel

2.1 Porównanie badań przedstawionych w rozdziale 2.4 32
3.1 Typy testów oprogramowania opisane w [50] 35
5.1 Średnia liczba testów podatności z wykorzystaniem automatycznych narzędzi wykonywana w skali roku (nie zawiera ręcznych testów penetracyjnych). Stan przed wdrożeniem systemu Mixeway. 67
5.2 Zasoby objęte testami podatności. Stan przed wdrożeniem Mixeway. 67
5.3 Średnia liczba testów podatności z wykorzystaniem automatycznych narzędzi wykonywana w skali roku (nie zawiera ręcznych testów penetracyjnych). Stan po wdrożeniu Mixeway. 68
5.4 Zasoby objęte testami podatności. Stan po wdrożeniu Mixeway. 69
6.1 Atrybuty zawarte w zestawie danych 73
6.2 Wartości pola kontekst 73
6.3 Znaczniki definiujące początek konkretnych pól w zestawie danych treningowych 76
6.4 Wybrane hiperparametry wykorzystywane w algorytmie SVM [117] 77
6.5 Wybrane hiperparametry wykorzystywane w algorytmie NN [118] 78
6.6 Wybrane hiperparametry wykorzystywane w algorytmie Lasu Losowego [120] 79
7.1 Wyniki porównania skuteczności wybranych algorytmów ML w klasyfikacji podatności bezpieczeństwa 82
7.2 Poziom TP, FP, FN i TN dla analizowanych algorytmów uczenia maszynowego 83
7.3 Miesięczna liczba weryfikacji bramki bezpieczeństwa oraz stosunek wdrożeń niespełniających wymagań bezpieczeństwa 92

Spis rysunków

1.1 Relatywny koszt usunięcia wykrytej podatności w zależności etapu, na którym znajduję się projekt. Źródło: National Institute of Standards and Technology [7] . . . 22
2.1 Obszary systemów bezpieczeństwa wraz z funkcjami pozwalającymi na orkiestrację (na podstawie [31]) . . . 28
3.1 Podstawowa forma procesu SDLC . . . 34
3.2 Zwinny model wytwarzania oprogramowania, źródło: [59] . . . 37
3.3 Łańcuch dostarczania oprogramowania . . . 38
3.4 Łańcuch dostarczania oprogramowania zawierający elementy bezpieczeństwa . . . 40
4.1 Przykładowe drzewo decyzyjne dla problemu “Czy podatność wymaga naprawy” . . . 44
4.2 Najpopularniejsze funkcje aktywacji. Od lewej - funkcja liniowa, binarna funkcja schodkowa, funkcja sigmoidalna, funkcja RELU . . . 45
4.3 Przykład sieci zbudowanej z jednego neuronu . . . 45
4.4 Przykład rekurencyjnej sieci zbudowanej z jednego neurona . . . 46
4.5 Przykład wyznaczenia maksymalnego marginesu między różnymi zbiorami punktów. . . . 48
4.6 Sposób działania tokenizacji na przykładzie pola zawierającego nazwę podatności . . . 50
5.1 Elementy realizujące łańcuch dostarczania oprogramowania przedstawione na Rys. 3.4 z wykorzystaniem prezentowanego rozwiązania 51
5.2 Niskopoziomowa architektura systemu Mixeway . . . 53
5.3 Definicja API wykonującego operacje na danym projekcie wewnątrz Mixeway 54
5.4 Sposób działania modułu wykrywania zasobów . . . 55
5.5 Sposób działania modułu zarządzania skanami bezpieczeństwa . . . 56
5.6 Sposób działania modułu zarządzania skanami bezpieczeństwa . . . 57
5.7 Interfejs zawierający metody pozwalające na integrację ze skanerami podatności typu SAST . . . 58
5.8 Opis działania bramki bezpieczeństwa, która weryfikuje zgodność aplikacji z polityką bezpieczeństwa . . . 60
5.9 Harmonogram wdrożenia Mixeway . . . 60
5.10 Kroki realizowane zgodnie z przyjętą procedurą zarządzania podatnościami 61
5.11 Umiejscowienie różnych skanerów podatności w infrastrukturze organizacji 63
5.12 Wynik działania mechanizmu Gitlab-CI w przypadku braku spełnienia polityki bezpieczeństwa . . . 64
5.13 Zrzut ekranu z głównego menu Mixeway . . . 65
5.14 Zrzut ekranu prezentującego konkretny projekt (część pierwsza) . . . 66

5.15 Zrzut ekranu prezentującego konkretny projekt (część druga) 66
6.1 Etapy metodyki badań eksperymentalnych 71
6.2 Podział zestawu danych na podatności potwierdzone i niepotwierdzone (cztery wykresy z góry) oraz liczba wykrytych podatności w podziale na źródła (wykres na dole). 72
6.3 Etapy, z których składa się przetwarzanie wstępne w prowadzonych badaniach eksperymentalnych 75
6.4 Zależność wartości Funkcji Straty od liczby warstw ukrytych dla różnych algorytmów 78
6.5 Zależność wartości metryki dokładności od liczby drzew w Lesie Losowym 79
7.1 Wartości metryk czułości i precyzji dla różnych wariantów Sieci Neuronowych 83
7.2 Wartości metryk uzyskane przez algorytm CNN: (a) Dokładność, (b) Precyzja 84
7.3 Wartości metryk uzyskane przez algorytm CNN: (a) Funkcja Straty, (b) Czułość 84
7.4 Wartości metryk uzyskanych przez algorytm NN (prosta wersja): (a) Dokładność, (b) Precyzja 85
7.5 Wartości metryk uzyskanych przez algorytm NN (prosta wersja): (a) Funkcja Straty, (b) Czułość 85
7.6 Wartości metryk uzyskanych przez algorytm RNN LTSM: (a) Dokładność, (b) Precyzja 86
7.7 Wartości metryk uzyskanych przez algorytm RNN LTSM: (a) Funkcja Straty, (b) Czułość 86
7.8 Wartości metryk uzyskanych przez algorytm RNN GRU: (a) Dokładność, (b) Precyzja 87
7.9 Wartości metryk uzyskanych przez algorytm RNN GRU: (a) Funkcja Straty, (b) Czułość 87
7.10 Wartości metryk precyzji oraz czułości w zależności od epoki dla modelu zbudowanego na podstawie zmanipulowanych danych 88
7.11 Średnia liczba podatności wykryta we wszystkich systemach w danym miesiącu 90
7.12 Konfiguracja bramki bezpieczeństwa w oprogramowaniu Mixeway z wykorzystaniem aplikacji VulnAuditor, która implementuje opracowany model 90
7.13 Liczba odpytań bramki bezpieczeństwa w stosunku dziennym (lewy wykres) oraz stosunek informacji o akceptacji i odrzuceniu zmiany ze względu na bezpieczeństwo (prawy wykres) 91
7.14 Miesięczna liczba weryfikacji bramki bezpieczeństwa z podziałem na spełnienie polityki bezpieczeństwa oraz jej złamanie 91

1. Wprowadzenie

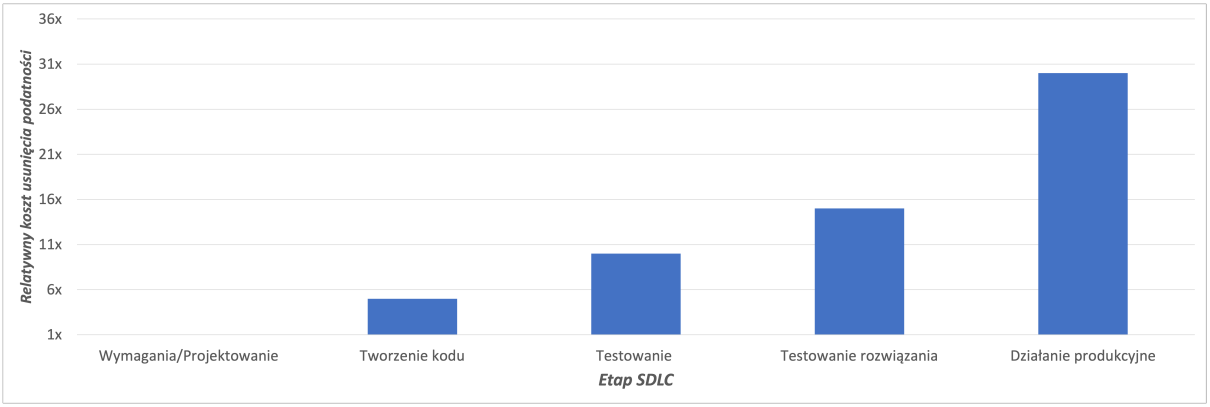
1.1. Wstęp

Na bezpieczeństwo systemu IT wpływa wiele czynników. Obszary takie jak odpowiedni podział uprawnień [1], bezpieczeństwo dostępu fizycznego do infrastruktury [2] czy procesy związane z reagowaniem na incydenty wykrywane dzięki mechanizmom monitorowania zdarzeń [3] są w znacznym stopniu odpowiedzialne za zminimalizowanie możliwości przeprowadzenia skutecznego ataku teleinformatycznego. Jednakże z raportu firmy Verizon, która przedstawiła wielopłaszczyznową analizę cyberataków w 2019 roku [4] wynika, że ponad 50% zdarzeń, które doprowadziły do wycieku informacji lub nieuprawnionego dostępu (do systemu lub infrastruktury organizacji) było możliwe dzięki wykorzystaniu luki w procesach biznesowych lub aplikacji. W tym zbiorze danych źródłem włamania najczęściej (ponad 60%) są niezabezpieczone aplikacje internetowe.

Z informacji przedstawionych powyżej można wywnioskować, że usprawnianie mechanizmów weryfikujących bezpieczeństwo aplikacji i aplikacji internetowych może znacząco wpłynąć na bezpieczeństwo organizacji. Mechanizmy te powinny umożliwiać wczesne wykrywanie podatności bezpieczeństwa. W kontekście bezpieczeństwa oprogramowania podatność rozumiemy jako *specyficzną wadę, przeoczenie lub błąd w oprogramowaniu, który umożliwia cyberprzestępcy wykonanie złośliwej operacji takiej jak: ujawnienie wrażliwych informacji, wpłynięcie na ciągłość działania systemu lub uzyskanie nieuprawnionego dostępu do aplikacji czy systemu operacyjnego* [5]. Na potrzeby niniejszej rozprawy podatność bezpieczeństwa będzie wymiennie określana mianem błędu oraz defektu.

Liczne analizy biznesowe pokazują, że z finansowego punktu widzenia koszt usunięcia podatności bezpieczeństwa różni się w zależności od tego na którym etapie cyklu życia systemu (ang. Software Development Life Cycle - SDLC) błąd zostanie wykryty [6]. Wspomniany wcześniej proces składa się z takich etapów jak: projektowanie, tworzenie kodu, testowanie oraz uruchomienie rozwiązania klientowi końcowemu. Rys. 1.1 przedstawia jak zmienia się relatywny koszt usunięcia defektu w zależności od fazy SDLC. Wykazuje on, że koszt usunięcia błędu wykrytego na etapie programowania jest dziesięciokrotnie mniejszy niż w momencie, w którym rozwiązanie zostało już w pełni wdrożone. Prowadzi to do wniosku, że mechanizmy bezpieczeństwa, które mają na celu wykrywanie podatności bezpieczeństwa w wytwarzanym oprogramowaniu, powinny skupiać się na początkowych etapach cyklu życia systemu.

Realizacja tego celu może okazać się bardzo trudnym zadaniem, głównie z uwagi na dynamikę w rozwoju narzędzi i procesów dostarczania oprogramowania. Obecnie większość operacji, które jeszcze kilka lat temu były wykonywane przez wyspecjalizowanych pracowników IT, realizowana jest w sposób automatyczny. Dzięki adaptacji mechanizmów umożliwiających ciągłe wdrożenie (ang. Continuous Deployment - CD), możliwe jest



Rys. 1.1: Relatywny koszt usunięcia wykrytej podatności w zależności etapu, na którym znajduję się projekt. Źródło: National Institute of Standards and Technology [7]

zminimalizowanie czasu, w którym klienci końcowi uzyskują dostęp do nowej wersji rozwijanego oprogramowania. Już w latach 2011-2013 organizacje takie jak: Facebook[1] [8] czy IMVU[2] [9] informowały o tym, że są w stanie wykonać od kilku do kilkudziesięciu wdrożeń dziennie dzięki rozwiązaniom takim jak CD. Z biznesowego punktu widzenia implementacja technik umożliwiających automatyzację łańcucha dostarczania oprogramowania przynosi same korzyści. W rzeczywistości natomiast żadna organizacja nie dysponuje zapleczem w postaci ekspertów bezpieczeństwa, którzy mogliby uczestniczyć w tak rozbudowanym i skomplikowanym procesie oraz weryfikować proponowane zmiany w momencie pojawienia się nowej wersji oprogramowania. Wykorzystanie narzędzi, które automatycznie weryfikują bezpieczeństwo aplikacji, działających zarówno statycznie, tj. weryfikujących kod źródłowy rozwiązania (ang. Static Application Security Testing - SAST), jak i dynamicznie, poprzez sprawdzenie możliwości wykorzystania konkretnych błędów na działającej aplikacji (ang. Dynamic Application Security Testing - DAST) jest konieczne, aby nie spowodować znacznego wydłużenia czasu dostarczenia nowej wersji rozwiązania i w tym samym czasie zapewnić bezpieczeństwo wdrożenia. Rezultatem działania automatycznych narzędzi bezpieczeństwa aplikacji są raporty. Generowane są one na podstawie wyniku testów podatności (SAST i DAST) i zawierają różnego rodzaju dane charakterystyczne dla przyjętego rozwiązania realizującego testy podatności. Istnieje jednak pewna część wspólna, która widoczna jest w raporcie generowanym przez większość narzędzi, na przykład nazwa podatności oraz jej opis. Innym przykładem jest krytyczność błędu, bardzo często opisywana jako CVSS (ang. Common Vulnerability Scoring System) [10]. Pierwsza wersja systemu klasyfikacji podatności została przedstawiona w 2005 roku. Najnowsza wersja standardu - CVSSv3.1 została opublikowana w czerwcu 2019 roku. CVSS przyjmuje wartości od 1 do 10, gdzie 0 przypisywana jest niegroźnym podatnościom natomiast 10 najbardziej krytycznym. Na określenie tego wyniku wpływają

[1] https://facebook.com
[2] https://imvu.com

takie składowe jak: wpływ na ciągłość działania oprogramowania w kontekście, którego podatność została wykryta (ang. Impact) oraz łatwość wykorzystania (ang. Exploitability). Wzór określający wynik CVSS dla konkretnej podatności wygląda następująco: $CVSS = ((0,6 * Impact) + (0,4 * Exploitability)) * f(Impact)$ gdzie $f(Impact) = 0$, jeśli $Impact = 0$ oraz $1,176$ w pozostałych przypadkach.

Niestety skanery podatności SAST jak i DAST z uwagi na sposób działania wykrywają podatności, które mogą być:

- *Błędnie sklasyfikowane* (ang. False Positive - FP) - błędy, które zostały wykryte przez automatyczne mechanizmy, w rzeczywistości nie występują w badanym oprogramowaniu. Błędnie nie sklasyfikowane podatności (ang. False Negative) choć wprowadzają dodatkowe ryzyko związane z brakiem zgłoszenia to nie wpływają negatywnie na proces dostarczania oprogramowania.
- *Poprawnie sklasyfikowane,* ale nie są istotne w kontekście badanego oprogramowania
- *Poprawnie sklasyfikowane, ale określona krytyczność jest nieodpowiednia* - W kontekście konkretnej podatności (sposób wykorzystania oprogramowania oraz użytkownika końcowego) krytyczność zgłoszona przez oprogramowanie skanujące może nie odzwierciedlać realnego zagrożenia.

Wymienione powyżej sytuacje powodują, że proces dostarczania oprogramowania, uwzględniający automatyczne testy bezpieczeństwa jest skomplikowany i trudny do wdrożenia. Sytuacja, w której spora część wdrożeń zostanie niezrealizowana przez mechanizmy bezpieczeństwa zgłaszające podatności, które po analizie okazały się fałszywie pozytywne (FP), jest nieakceptowalna z perspektywy biznesowej.

Przedmiotem niniejszej rozprawy jest analiza wykorzystania mechanizmów uczenia maszynowego (ang. Machine Learning - ML) do klasyfikacji podatności bezpieczeństwa w oprogramowaniu. Dane wejściowe, które będą brane pod uwagę tworzą zbiór błędów wykrytych w głównej mierze przez skanery DAST i SAST. Stworzony zestaw danych zostanie ręcznie podzielony na dwie grupy - podatności, które są potwierdzone i powinny być poprawione (ang. Confirmed and Relevant Vulnerability - CRV) oraz wykryte, ale nie istotne (and. Detected but not Relevant Vulnerability - DNRV). Do klasyfikacji zostaną wykorzystane także metody związane z przetwarzaniem języka naturalnego (ang. Natural Language Processing - NLP). Tak przygotowany klasyfikator zostanie wdrożony w środowisku sieci operatora telekomunikacyjnego jako część systemu, odpowiadającego za orkiestrację narzędzi bezpieczeństwa w rzeczywistym łańcuchu dostarczania oprogramowania.

W rezultacie badań prowadzonych w ramach niniejszej rozprawy powstał system Mixeway, który jest udostępniony na mocy licencji GPL-3.0 za pośrednictwem serwisu

GitHub[3] (kod źródłowy) oraz DockerHub[4] (gotowy obraz aplikacji). W momencie tworzenia niniejszej pracy oprogramowanie zostało pobrane ponad 14 000 razy. Mixeway jest wdrożony oraz operacyjnie wykorzystywany w procesach związanych z zapewnianiem bezpieczeństwa w łańcuchu dostarczania oprogramowania w infrastrukturze operatora telekomunikacyjnego Orange Polska, gdzie został wyróżniony dwoma nagrodami, zarówno na poziomie lokalnym jak i globalnym (Grupa Orange). Wyniki badań, które zostały wykorzystane do opracowania wspomnianego oprogramowania zostały przedstawione w pracy *"Context-Aware Software Vulnerability Classification Using Machine Learning", 2021* [11] oraz zaprezentowane na branżowych konferencjach: PLNOG19, What The H@ck 2019, Advanced Threat Summit 2020. Dodatkowo w kwietniu 2020 ukazał się artykuł opisujący opracowywane rozwiązanie zatytułowany *"Orange Funds New CI/CD Security Tool"* opublikowany w **HardenStance briefing**.

1.2. Cele i teza rozprawy

Niniejsza rozprawa ma cztery podstawowe cele:

- Eksperymentalne porównanie dokładności algorytmów uczenia maszynowego pod kątem możliwości ich wykorzystania w celu wsparcia procesów związanych z zapewnieniem bezpieczeństwa w łańcuchu dostarczania oprogramowania.
- Implementacja rozwiązania, które orkiestruje pracę narzędzi bezpieczeństwa takich jak skanery podatności oraz wykorzystuje implementację wybranego klasyfikatora w celu wsparcia procesu decyzyjnego dotyczącego wdrożenia oprogramowania.
- Wdrożenie rozwiązania w rzeczywistej sieci operatora telekomunikacyjnego.
- Udowodnienie skuteczności wdrożonego rozwiązania bazując na danych historycznych.

Realizacja powyższych celów pozwala sformułować i obronić następującą tezę rozprawy:

- Możliwe jest stworzenie i wdrożenie w sieci operatora telekomunikacyjnego efektywnego systemu teleinformatycznego, który pełni rolę orkiestratora narzędzi bezpieczeństwa oraz wykorzystuje sieci neuronowe w celu określenia, które z wykrytych podatności są konieczne do poprawy przez zespoły programistyczne, co w rezultacie spowoduje wzrost liczby przeprowadzonych testów bezpieczeństwa oraz poprawi poziom zabezpieczeń projektu, w ramach którego zostanie uruchomiony.

1.3. Układ pracy

Układ niniejszej rozprawy jest następujący: w Rozdziale 2 przedstawione zostały wyniki najnowszych badań w dziedzinie cyberbezpieczeństwa ze szczególnym uwzględnieniem metod wykorzystujących algorytmy uczenia maszynowego. Rozdział 3 zawiera opis metod

[3] https://github.com/mixeway/mixewayhub
[4] https://hub.docker.com/u/mixeway

prowadzenia projektów teleinformatycznych wraz ze zdefiniowaniem problemu zapewnienia bezpieczeństwa w automatycznym łańcuchu dostarczenia oprogramowania. Rozdział 4 zawiera opis wykorzystanych algorytmów ML. Następnie, w Rozdziale 5 przedstawiono rozwiązanie Mixeway, które wykorzystano do przeprowadzenia badań eksperymentalnych opisanych w Rozdziale 6. Wyniki przeprowadzonych analiz przedstawione zostały z kolei w Rozdziale 7 natomiast Rozdział 8 zawiera wnioski końcowe oraz wskazuje potencjalne kierunki dalszych badań.

2. Przegląd istniejących rozwiązań

W ostatnich latach w świecie IT połączenie terminów takich jak: cyberbezpieczeństwo oraz uczenie maszynowe cieszy się coraz większym zainteresowaniem wśród badaczy [12]. W całej dziedzinie bezpieczeństwa IT warto wyróżnić szczególnie dwa obszary, w których sposoby na wykorzystanie sztucznej inteligencji są na bardzo zaawansowanym etapie – bezpieczeństwo sieciowe [13] oraz bezpieczeństwo aplikacji internetowych [14].

Reagowanie na incydenty bezpieczeństwa, które zostały wykryte dzięki mechanizmom monitorowania zdarzeń w warstwie sieciowej modelu ISO/OSI [15] jest bardzo istotne z punktu widzenia zarządzania bezpieczeństwem w każdej organizacji [16]. Możliwość wykrycia cyberataku w warstwie sieciowej przez bardzo długi czas opierała się na definicji metryk i przygotowaniu odpowiednich scenariuszy, których wystąpienie w weryfikowanej próbce ruchu sieciowego świadczyło o zaobserwowaniu podejrzanego zachowania lub ataku. Wykorzystanie do tego celu technik uczenia maszynowego znacznie zwiększa możliwości omawianego procesu. Wybrane prace w tym zakresie zostały przedstawione w Rozdziale 2.1.

Bezpieczeństwo aplikacji już od dłuższego czasu jest jednym z głównych elementów, które składają się na jakość oprogramowania. Organizacje, aby utrzymywać jakość wytwarzanych rozwiązań korzystają z coraz większej liczby narzędzi, które wspierają procesy takie jak: zarządzanie podatnościami, automatyzacja operacji związanych z testowaniem aplikacji czy śledzenie zmian w wytwarzanym kodzie źródłowym. W celu optymalizacji posiadanych zasobów przeprowadzono wiele badań związanych ze sposobami integracji oraz orkiestracji narzędzi bezpieczeństwa (co przedstawiono w Rozdziale 2.2). Zgodnie z obecnym stanem wiedzy istnieje kilka rozwiązań, które demonstrują wykorzystanie algorytmów ML w celu analizy podatności bezpieczeństwa wykrytych w oprogramowaniu. Prace te można podzielić na dwie grupy – wykrywanie podatności bezpieczeństwa (Rozdział 2.3) oraz klasyfikacja wykrytych defektów (Rozdział 2.4).

2.1. Wykrywanie ataków sieciowych w oparciu o mechanizmy uczenia maszynowego

Aplikacje wykorzystujące algorytmy ML w domenie bezpieczeństwa sieciowego mogą być sklasyfikowane do następujących kategorii (podział zaproponowany w [17]): systemy wykrywania anomalii/włamań oraz ochrona urządzeń końcowych.

2.1.1. Systemy wykrywania włamań

Wykrywanie anomalii w próbkach ruchu sieciowego, a następnie wykorzystywanie tego typu podziału w implementacji systemów wykrywania włamań (ang. Intrusion Detection System - IDS) jest obecnie popularnym zagadnieniem wśród badaczy zajmujących się bezpieczeństwem [18]. Zdecydowana większość systemów typu IDS, które nie korzystają z mechanizmów ML bazuje na sygnaturach. Oznacza to, że wykryte zostaną ataki, których sygnatura znajduje się we wcześniej przygotowanej bazie danych zawierające

odpowiednie wzorce. Niewątpliwą zaletą tego typu mechanizmu jest brak wykryć fałszywie pozytywnych - każdy alarm oznacza incydent bezpieczeństwa. Niestety podejście takie uniemożliwia reakcję na ataki typu 0-day (czyli takich, których sygnatury nie są dostępne) [19] [20]. Rozwiązaniem tego typu problemów są metody detekcji anomalii, które korzystają z algorytmów ML. Dzięki temu możliwe jest wykrywanie także nowych typów ataków. Należy jednak zwrócić uwagę, że takie podejście może powodować generowanie dużej liczby alarmów, które są fałszywie pozytywne (są wykrywane przez algorytm jednak po analizie nie świadczą o wystąpieniu incydentu bezpieczeństwa) [20].

Badacze w pracy [21] przeprowadzili eksperyment mający na celu porównania skuteczności technik uczenia nadzorowanego (klasyfikacja) i uczenia nienadzorowanego (grupowanie) do wykrywania podejrzanej aktywności w próbce ruchu sieciowego. W przypadku wykrywania znanych ataków, najlepszą skutecznością wykazał się algorytm drzewa decyzyjnego (uczenie z nadzorem). Z kolei dla ataków, które nie znajdowały się w próbce danych treningowych nie zaobserwowano znaczącej różnicy w skuteczności technik uczenia z nadzorem i bez. Model *HELAD* przedstawiony w pracy [22] wykorzystuje hybrydową wersję sieci neuronowych w celu wykrywania anomalii bezpieczeństwa w ruchu sieciowym. Zaprezentowane rozwiązanie uzyskało precyzję wykrycia ataków w badanej próbce na poziomie 99%, co udowadnia skuteczność opracowanej metody.

W kontekście wykrywania zagrożeń w ruchu sieciowym realizowanym na potrzeby urządzeń Internetu Rzeczy (ang. Internet of Things - IoT) istnieje wiele badań, przedstawiających użyteczność algorytmów ML w celu implementacji systemu klasy IDS. Badania zaprezentowane w [23] przedstawiają klasyfikator zbudowany z wykorzystaniem algorytmu Maszyn Wektorów Nośnych (ang. Support Vector Machine - SVM), pozwalającym wykrywać ataki typu DDoS (ang. Distributed Denial of Service). Wyniki uzyskane w [24] udowadniają skuteczność algorytmów takich jak: Regresja Logiczna, Drzewa Decyzyjne, SVM oraz Sieci Neuronowe w celu detekcji ataków i anomalii w ruchu sieciowym generowanym przez urządzenia IoT.

Urządzenia przenośne stają się coraz bardziej dominującym narzędziem wśród ludzi do komunikacji i do przechowywania wrażliwych informacji [25]. Badania zaprezentowane w [26] przedstawiają rozwiązanie, które umożliwia identyfikację zagrożeń cybernetycznych w sieciach mobilnych 5G przy użyciu samoadaptującego się systemu opartego na głębokim uczeniu. Prototyp implementuje dwa moduły - ASD (wykrywanie symptomów anomalii) zbudowanego z wykorzystaniem DBF & SAE (ang. Deep Belief Network and Stacked Auto-Encoders) oraz NAD (wykrywanie anomalii sieciowych) zbudowanego z wykorzystaniem rekurencyjnych sieci neuronowych.

2.1.2. Ochrona urządzeń końcowych

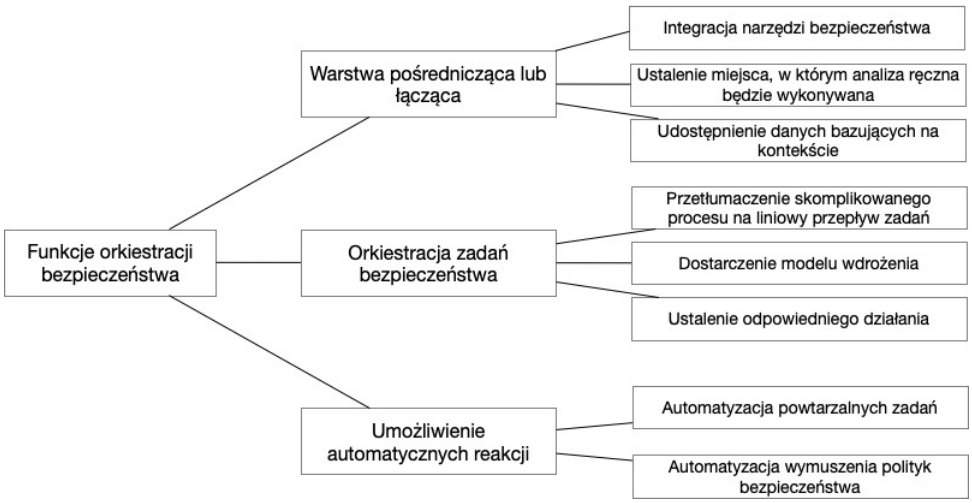
Innym sposobem wykorzystania analizy ruchu sieciowego są mechanizmy ochrony urządzeń końcowych takich jak: komputery, tablety czy telefony komórkowe [27]. W większości przypadków złośliwe oprogramowanie, aby funkcjonować, potrzebuje komu-

nikacji z serwerami zarządzającymi (ang. Command and Control - C&C) [28]. Praca przedstawiona w [29] prezentuje badanie weryfikujące skuteczność wybranych algorytmów ML w celu wykrycia obecności złośliwego oprogramowania szyfrującego dysk twardy ofiary (ang. ransomware). Według opublikowanych wyników najlepszy rezultat uzyskał algorytm drzew decyzyjnych. Rozbudowane badania dotyczące wykrywania złośliwych aplikacji, przeznaczonych na system operacyjny Android zostały opublikowane w [30]. Prezentowany prototyp *DL-DRIOD* wykorzystując głęboką sieć neuronową osiągnął wyniki na poziomie 98% dokładności wykrycia niebezpiecznych aplikacji przeznaczonych na urządzenia mobilne.

2.2. Orkiestracja narzędzi bezpieczeństwa

Organizacje, które posiadają rozbudowane zespoły zajmujące się bezpieczeństwem IT, dysponują wieloma różnego rodzaju narzędziami wykorzystywanymi podczas codziennej pracy. W obszarach takich jak: monitorowanie zdarzeń, obsługa incydentów, czy zarządzanie podatnościami zespoły takie dysponują kilkoma (lub kilkunastoma) rozwiązaniami. Orkiestracja pozwala na integrację narzędzi, które służą do podobnego celu, przez co zarówno korzystanie, jak i utrzymywanie ekosystemu jest dużo prostsze.

Badacze w pracy [31] obszernie analizują i klasyfikują rozwiązania służące do orkiestracji narzędzi bezpieczeństwa. Prezentowane rozwiązanie, które jest opisywane w niniejszej



Rys. 2.1: Obszary systemów bezpieczeństwa wraz z funkcjami pozwalającymi na orkiestrację (na podstawie [31])

rozprawie, należy do kategorii "Integracja narzędzi bezpieczeństwa" (Rys. 2.1). Rozwiązania takie jak [32] oraz [33] są komercyjnymi systemami służącymi do orkiestracji zadań związanych z obsługą incydentów, pozwalającymi na elastyczne projektowanie procesów związanych z mitygacją zagrożeń. Praca opisana w [34] także przedstawia rozwiązanie, które pozwala na integrację narzędzi służących do przetwarzania informacji o cyberata-

kach, bazując na danych pochodzących z różnego rodzaju systemów realizujących zadania związane z rozpoznawaniem zagrożeń (ang. Threat Intelligence - TI).

Narzędzia orkiestrujące procesy związane z zarządzaniem podatnościami [35] i [36] posiadają funkcje takie jak: przechowywanie historii o podatnościach, umożliwiają automatyczne pobieranie informacji o błędach ze źródła wykrycia (np. skaner podatności) oraz umożliwiają modelowanie procesu związanego ze zgłaszaniem błędów do programistów w celu ich usunięcia.

2.3. Wykrywanie podatności w oprogramowaniu w oparciu o algorytmy uczenia maszynowego

W tym podrozdziale omówiono istniejące badania oraz prototypy oprogramowania, które wykorzystują algorytmy ML w celu wykrycia podatności bezpieczeństwa w kodzie źródłowym badanej aplikacji.

Praca opisana w [37] przedstawia skuteczność wykorzystania sieci neuronowych w celu zidentyfikowania miejsca w kodzie źródłowym, w którym występuje podatność bezpieczeństwa (dotyczy aplikacji stworzonych w języku C/C++). Badania przedstawione w [38] potwierdzają otrzymane powyżej wyniki. Przedstawione rozwiązanie *SySeVR* poprzez wykorzystanie zdecydowanie większego zestawu danych służących jako dane wejściowe opracowanego modelu umożliwiły uzyskanie dokładności na poziomie 90%. W wyniku dalszych badań autorom udało się wykryć 15 nieznanych oraz nieopublikowanych wcześniej podatności bezpieczeństwa w projektach, które są umieszczone w serwisie GitHub[5]. Obie prace skupiają się na weryfikacji skuteczności modelu zbudowanego z wykorzystaniem sieci neuronowych.

Praca opisana w [39] analizuje możliwość wykorzystania algorytmu SVM. Otrzymane wyniki potwierdzają to, że algorytm SVM może być wykorzystywany w celu wykrywania podatności bezpieczeństwa w kodzie źródłowym. Niestety zestaw danych treningowych został ograniczony do jednej aplikacji stworzonej w języku Java, przez co można podejrzewać, że opracowany model nie jest dostatecznie wiarygodny, a wyniki uzyskane podczas analizy aplikacji o innym charakterze mogą zdecydowanie odbiegać od oczekiwanych.

Badania przedstawione w [40] skupiają się na analizie różnych algorytmów, pozwalających ocenić ich przydatność w operacjach związanych z wykrywaniem podatności w kodzie źródłowym napisanym w języku C/C++. Otrzymane wyniki pokazują, że największą skuteczność uzyskał model zbudowany z wykorzystaniem algorytmu Konwolucyjnych Sieci Neuronowych (ang. Convolutional Neural Network - CNN).

[5] https://github.com

2.4. Klasyfikacja wykrytych podatności z wykorzystaniem algorytmów uczenia maszynowego

Poniżej przedstawiono badania, które mają na celu stworzenie klasyfikatora pozwalającego na grupowanie wykrytych (przez różne źródła danych) podatności bezpieczeństwa.

Praca przedstawiona w [41] weryfikuje możliwość wykorzystania algorytmów takich jak: Las Losowy (ang. Random Forrest - RF), C4.5 drzewa decyzyjne, Naive Bayes i Regresja Logiczna (ang. Logistic Regression - LR) w celu klasyfikacji podatności. Autorzy zaproponowali rozwiązanie, które bazując na zgłoszonych błędach (w narzędziach służących do tego celu np. Bugzilla[6]) określa czy wykryta podatność może zostać skategoryzowana jako BV (Bohr Vulnerability - łatwa do wykrycia oraz prosta do usunięcia), czy MV (Mandel Vulnerability - trudna do wykrycia i skomplikowana do usunięcia). Bazując na wynikach uzyskanych podczas opisanego eksperymentu można stwierdzić, że algorytm RF uzyskał najlepsze wyniki. Kolejna praca, która wykorzystuje algorytmy ML do celu klasyfikacji podatności (określenia czy podatność należy do kategorii takiej jak np. Cross-Site Scripting) została przedstawiona w [42]. Eksperyment został przeprowadzony z wykorzystaniem danych uzyskanych z National Vulnerability Database (NVD), która posiada informacje na temat publicznie udostępnionych podatności bezpieczeństwa w różnego rodzaju oprogramowaniu. Przedstawione badania weryfikują skuteczność modelu zbudowanego z wykorzystaniem algorytmu SVM.

Kolejne badania, które wykorzystują bazę NVD opisano w [43]. Autorzy tej pracy prezentują model, który na podstawie opisu podatności bezpieczeństwa jest w stanie przewidzieć krytyczność błędu. Rozwiązanie to jest bardzo ciekawą alternatywą dla powszechnie wykorzystywanego indeksu CVSS. W przedstawionym eksperymencie algorytm NN uzyskał lepsze wyniki niż SVM. Badania przedstawione w [44] analizują podatności bezpieczeństwa, które zostały wykryte w różnego rodzaju oprogramowaniu (na przykład systemie operacyjnym Windows XP) pod kątem możliwości przewidzenia, ile błędów w konkretnym rozwiązaniu zostanie zgłoszonych w przyszłości. Rozwinięciem tej pracy są badania zaprezentowane w [45]. Autorom udało się zbudować model wykazujący się dokładnością na poziomie 70%. Obie prace wspomniane powyżej ([44] i [45]) jako źródło danych także wykorzystują bazę NVD. Prace [41] i [42] przedstawiają możliwości związane z wykorzystaniem algorytmów ML w celu klasyfikacji podatności bezpieczeństwa. Zestaw danych treningowych zawiera informacje dotyczące potwierdzonych podatności wykrytych w oprogramowaniu (pochodzących z systemów służących do zgłaszania błędów oraz bazy NVD). Udowodniono skuteczność przedstawionych modeli, jednakże żadne z wymienionych rozwiązań nie uwzględnia kontekstu aplikacji. Przykładem może być podatność opisana przez numer CVE-2017-7529, która opisuje podatność typu “Integer overflow” występującą w serwerze aplikacyjnym Nginx (dotyczy tego oprogramowania w wersjach 0.5.6 - 1.13.2). Podatność ta jest wykrywana przez narzędzia skanujące na pod-

[6] https://bugzilla.org

stawie banera (odpowiedniego nagłówka, który odsyła serwer w odpowiedzi na żądanie HTTP). Domyślnie baner zawiera informacje takie jak: nazwa i wersja serwera aplikacyjnego. Jeśli administrator zmieni wartość odpowiedniego nagłówka na jakikolwiek inny ciąg znaków to wykryta podatność, która zostanie zaraportowana, może być traktowana jako błędnie oceniona (ang. False Positive - FP). Bez kontekstu, w jakim działa aplikacja często niemożliwa jest klasyfikacja podatności w sposób odpowiadający wymaganiom organizacyjnej polityki bezpieczeństwa.

2.5. Rozwiązania komercyjne

Istnieją również rozwiązania komercyjne, służące do klasyfikacji podatności. Przykładem może być rozwiązanie "Audit Assistant"[7], które zostało uruchomione przez firmę MicroFocus. Omawiane oprogramowanie jako dane wejściowe wykorzystuje zestaw podatności wykrytych przez statyczny skaner kodu źródłowego MicroFocus Fortify[8] natomiast wynikiem jego działania jest informacja, czy dana podatność jest możliwa do wykorzystania (podatność może zostać wykorzystana np. do uzyskania nieautoryzowanego dostępu do systemu) lub jest błędnie wykryta (FP wykryty przez skaner Fortify). Niestety, producenci oprogramowania nie udostępniają informacji, dotyczącej wykorzystywanych algorytmów w swoich produktach.

2.6. Podsumowanie

Rozwiązania opisane w rozdziałach 2.3, 2.4 oraz 2.5 bazują na uczeniu z nadzorem (zestaw danych treningowych zawiera informacje o poprawnej kategorii). Tabela 2.1 przedstawia porównanie wyników opublikowanych badań z wdrażanym rozwiązaniem (Mixeway). Znaczne różnice w kategoriach takich jak źródło danych dla opracowanego klasyfikatora czy możliwość wykorzystania rozwiązania w procesie CICD są adresowane przez niniejszą rozprawe.

Analizując wyniki uzyskane przez cytowanych powyżej badaczy można jednoznacznie stwierdzić, że algorytmy ML mogą być wykorzystane w celu klasyfikacji podatności bezpieczeństwa. Klasyfikator przedstawiony w [41] może zostać wykorzystany jako rozwiązanie pozwalające na usprawnienie procesu, polegającego na określeniu priorytetów dla zespołów programistycznych przez wskazanie podatności typu BV, które mogą być usuwane w pierwszej kolejności, jako że ich eliminacja powinna być stosunkowo prosta.

Niestety żaden z przedstawionych modeli nie wykorzystuje informacji na temat kontekstu, w jakim działa weryfikowany system IT. Informacja taka jest szczególnie istotna w przypadku wykorzystania klasyfikatora jako źródła informacji dotyczącego tego, czy badany system spełnia politykę bezpieczeństwa organizacji. Dobrym przykładem może być podatność "SSL/TLS Untrusted Certificate", która jest wykrywana przez większość

[7] https://www.microfocus.com/media/data-sheet/fortify-audit-assistant-ds.pdf
[8] https://www.microfocus.com/en-us/solutions/application-security

Tabela 2.1: Porównanie badań przedstawionych w rozdziale 2.4

Badanie	Typ Klasyfikacji	Źródło danych	Możliwość wykorzystania w CICD
[41]	BV lub MV	Bug Trackers	Tak, z ograniczeniami
[42]	Typ podatności	NVD	Nie
[43]	Krytyczność podatności	NVD	Nie
[44]	Liczba podatności w przyszłości	NVD	Nie
[45]	Czas do publikacji nowej podatności	NVD	Nie
Audit Assistant	Potwierdzone i niepotwierdzone podatności	Fortify	Tak
Prezentowane rozwiązanie (Mixeway)	Potwierdzone i niepotwierdzone podatności	Dowolne	Tak

skanerów podatności. Za każdym razem, gdy podatność tego typu zostanie wykryta w aplikacji zostanie ona oznaczona jako BV, podczas gdy w niektórych przypadkach jej występowanie może naruszać politykę bezpieczeństwa (w sytuacji, gdy z danej aplikacji korzystają klienci organizacji) a w innych nie (w przypadku, gdy aplikacja jest wykorzystywana wewnętrznie przez pracowników firmy lub dostęp następuje przez elementy takie jak: proxy czy system równoważenia obciążenia (ang. Load Balancing - LB), gdzie realizowana jest funkcja odszyfrowywania i ponownego szyfrowania ruchu sieciowego). Na drugim biegunie sytuują się rozwiązania komercyjne takie jak: “Audit Assistant” (rozdział 2.5), które potrzebują do odpowiedniego działania poprawnie przygotowanych danych wejściowych. W tym konkretnym przypadku rozwiązanie zadziała tylko dla podatności wykrytych przez oprogramowanie MicroFocus Fortify, co jest bardzo dużym ograniczeniem (Tabela 2.1).

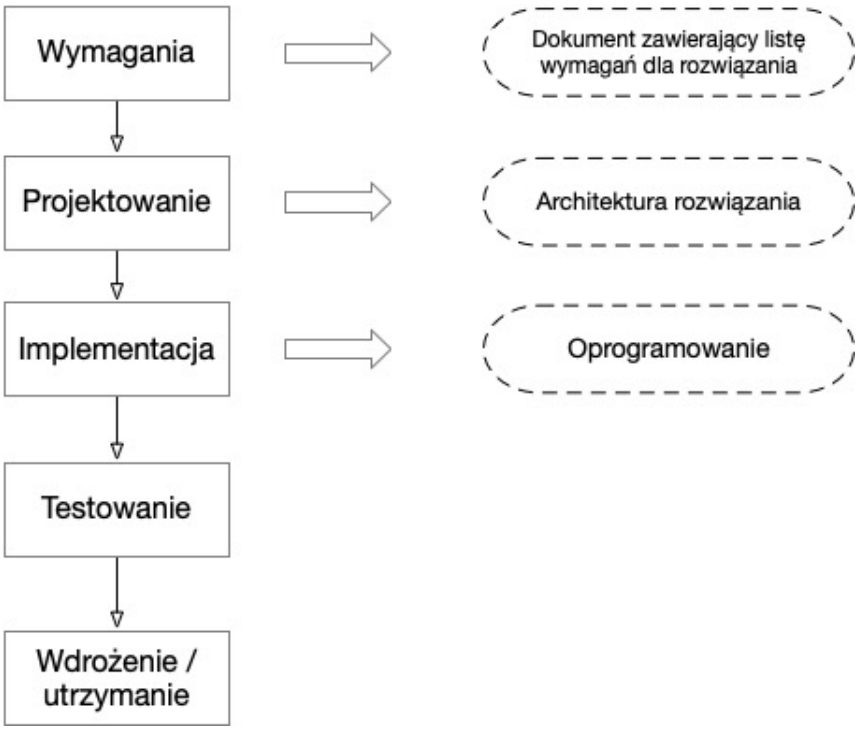
W odróżnieniu od opisanych badań przedstawione w tej pracy rozwiązanie wykorzystuje potencjalne i niepotwierdzone podatności bezpieczeństwa, które zostały zebrane przy pomocy automatycznych narzędzi tj. skanerów podatności. Następnie dane te, po uprzednim przetworzeniu i przygotowaniu, są przesyłane do modelu przygotowanego z wykorzystaniem algorytmu NN z wykorzystaniem technik przetwarzania języka naturalnego (ang. Natural Language Processing - NLP). W tej dziedzinie typowo algorytmy NN uzyskują najlepsze wyniki [46]. Klasyfikator określa, czy podatność należy do jednej z dwóch klas - potwierdzona i niepotwierdzona. Daną wejściową może być każda podatność bezpieczeństwa odpowiednio wcześniej przetworzona (bez ograniczenia co do typu podatności czy sposobu wykrycia). W wyniku weryfikowana podatność zostanie sklasyfikowana i oznaczona jako taka, która powinna zostać usunięta lub nie, bazując na danych takich jak: opis podatności oraz kontekst aplikacji.

Dodatkowo, aby tak zbudowany proces był odpowiednio zautomatyzowany rozwiązanie musi mieć możliwość pełnej kontroli dotyczącej uruchamiania skanów podatności (na zintegrowanym oprogramowaniu). Takiej funkcji (zdalnego zarządzania testami podatności) nie posiadają obecnie dostępne rozwiązania ([35], [36]).

3. Metodyki prowadzenia projektów IT

3.1. Cykl życia projektu

Termin cyklu życia projektu (ang. Software Development Life Cycle - SDLC) powstał w latach 60 XX wieku, ostatecznie został jednak zdefiniowany w 1971 roku [47]. SDLC jest procesem, który opisuje etapy wykonywane przez inżynierów podczas tworzenia, rozwoju i utrzymania rozwiązania. Rys. 3.1 przedstawia podstawowe etapy, które znajdują się w



Rys. 3.1: Podstawowa forma procesu SDLC

procesie SDLC:

- *Zebranie wymagań* (analiza) - etap ten polega na kompletnym zebraniu wymagań oraz funkcjonalności i sposobu, w jaki mają być one realizowane. Etap ten jest bardzo istotny, ponieważ jest pierwszą operacją w procesie i od jego wyniku zależy kompletność kolejnych kroków, które są realizowane w projekcie.
- *Projektowanie* - krok ten polega na stworzeniu architektury rozwiązania. Elementy takie jak: sposób integracji tworzonego oprogramowania z innymi systemami lub aplikacjami, technologia wykonania rozwiązania (np. wykorzystanie konkretnego języka programowania), czy określenie, zgodnie z jaką infrastrukturą będzie działał projekt są uwzględniane w opracowywanej koncepcji.
- *Implementacja* - etap tworzenia oprogramowania. Wynik tej fazy jest mocno powiązany z dwiema poprzednimi. Brak odpowiednio zdefiniowanych (lub niekompletnych) wymagań będzie skutkował powstaniem wadliwego produktu, natomiast

nieodpowiednio zaprojektowana architektura rozwiązania może wpływać np. na jego wydajność.

- *Testowanie* - etap, na którym podejmowana jest decyzja dotycząca tego, czy wytworzone rozwiązanie spełnia wymagania opisane i zdefiniowane na początku SDLC. Liczba i typy testów weryfikacyjnych są różne, a wykorzystanie konkretnego zestawu testów jest zależne od charakteru projektu. Lista typów testów została opisana w Tabeli 3.1.
- *Użytkowanie* (wdrożenie) - faza, w której system został już uruchomiony za pomocą docelowej infrastruktury produkcyjnej, która jest dostępna dla użytkowników końcowych.

W zależności od źródła [48] [49] można spotkać się z dodatkowymi etapami. Pierwszy z nich to analiza, na której wykonywana jest weryfikacja i możliwość implementacji rozwiązania (przed projektowaniem). Drugi dotyczy wycofania systemu na samym końcu. Etapy te jednak w odpowiednio zaprojektowanym procesie powinny zostać uwzględnione już w przedstawionych i zaprezentowanych fazach.

Tabela 3.1: Typy testów oprogramowania opisane w [50]

Nazwa testu	Opis
Testy jednostkowe (ang. Unit Testing)	Testy wykonywane na pojedynczych elementach programu takich jak metody lub procedury.
Testy integracyjne (ang. Integration Testing)	Testy wykonywane są na konkretnym interfejsie w celu oceny jego działania oraz oceny działania integracji z innymi systemami.
Testy systemu (ang. End-to-end Testing)	Testy dotyczą kompletnego rozwiązania. Emulowane jest zachowanie użytkownika końcowego w celu weryfikacji wszystkich interfejsów, które wchodzą w skład tworzonego oprogramowania.
Testy akceptacyjne (ang. Acceptance Testing)	Formalny zestaw testów, który weryfikuje określone wymagania, które zostały przedstawione na początkowym etapie tworzenia systemu.

Mimo że świat IT słynie z niespotykanego w innych branżach tempa, w którym rozwijają się narzędzia oraz procesy wytwarzania produktów, SDLC wydaje się procesem, który do dzisiaj jest wykorzystywany w niezmienionej niemal formie. Procesy takie jak S-SDLC (ang. Secure Software Development Life Cycle) [51],[52] czy STLC (ang. Software Testing Life Cycle) [53] bazują na SDLC. O ile sam proces nie uległ zmianom w ciągu ostatnich lat, można jednak zaobserwować różnicę w sposobie realizacji konkretnych kroków. Dzięki zwiększającej się dostępności oprogramowania, które dostosowane jest do wykorzystania na konkretnych etapach i mającemu na celu większe wsparcie zespołów projektowych, obserwujemy powstawanie nowych metodyk, które realizują SDLC. Najpopularniejszymi wśród nich są: model kaskadowy (ang. Waterfall) [54] oraz zwinny (ang. Agile) [55].

3.2. Model kaskadowy

Realizacja projektu, mającego na celu stworzenie systemu informatycznego z wykorzystaniem modelu kaskadowego, polega na sekwencyjnym wykonywaniu etapów, które są opisane w SDLC (Rys. 3.1).

Za realizację konkretnych etapów odpowiada osoba, pełniąca rolę kierownika projektu (Project Manager - PM), która koordynuje prace oraz ma za zadanie opracować następujące produkty: zakres, harmonogram oraz chronologię wykonywanych zadań [56]. Wspomniane elementy (na przykład harmonogram) są bardzo istotne w modelu kaskadowym. Błędy na tym etapie związane z niedoszacowaniem czasu na konkretny etap mogą prowadzić do opóźnień w realizacji projektu.

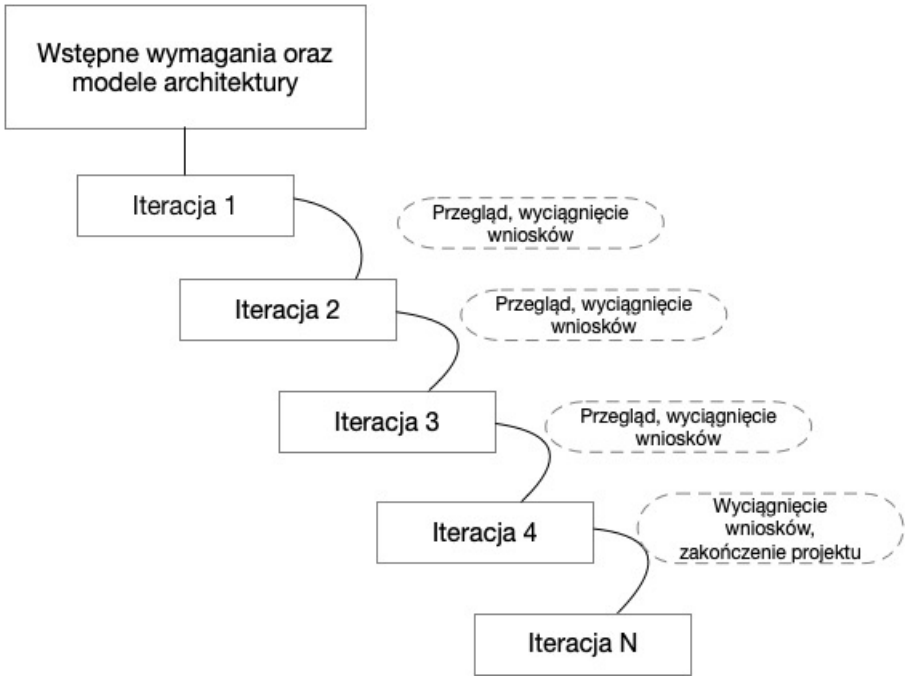
3.3. Model zwinny

Model kaskadowy (rozdział 3.2) w wielu organizacjach funkcjonuje bardzo wydajnie do dzisiaj. Niestety, sam w sobie zawiera wiele ograniczeń, które uniemożliwiają wdrażanie niewielkich zmian w oprogramowaniu w krótkim czasie. W 2001 roku zaproponowana została zatem zmiana w sposobie realizacji projektów informatycznych, która ma na celu pokonanie ograniczeń wprowadzanych przez metody kaskadowe [57]. Zaproponowane zmiany miały na celu poprawę jakości wytwarzanego oprogramowania, zwiększenie elastyczności procesu oraz skrócenie czasu potrzebnego do dostarczenia efektu prac klientom końcowym. Całość została przedstawiona jako 12 zasad oraz 4 wartości nazwane "Agile Manifesto". Wspomniane wartości [58] to:

- Ludzie i interakcje ponad procesami i narzędziami.
- Działające oprogramowanie ponad rozbudowaną dokumentacją.
- Współpraca z klientem ponad negocjacjami kontraktu.
- Reagowanie na zmiany ponad przyjętym planem.

Natomiast zasady [58] to:

- Wczesne i ciągłe dostarczanie oprogramowania.
- Gotowość na zmiany w wymaganiach dla lepszej współpracy z klientem.
- Częste dostarczanie poprawnie działającego oprogramowania.
- Codzienne interakcje z zespołem, pracującym nad rozwiązaniem (zarówno od strony biznesowej jak i technicznej).
- Tworzenie projektu wokół zmotywowanych jednostek.
- Komunikacja "twarzą w twarz".
- Najważniejszą miarą postępu jest działające oprogramowanie.
- Ciągły spokój.
- Ciągła uwaga na doskonaleniu technicznym.
- Prostota.
- Samo organizujące się zespoły.
- Regularne dyskusje wewnątrz zespołów dotyczące poprawy efektywności.



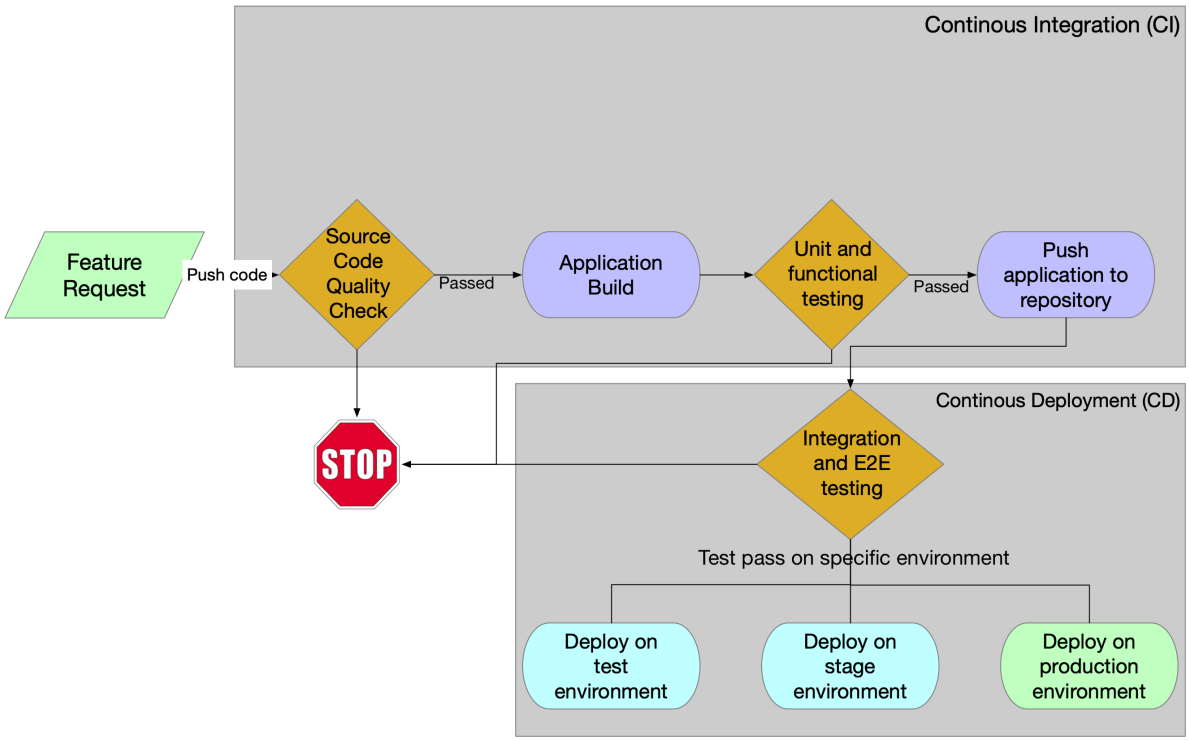
Rys. 3.2: Zwinny model wytwarzania oprogramowania, źródło: [59]

Etapy realizacji projektu, bazującego na zwinnych metodykach prowadzenia projektu przedstawione zostały na Rys. 3.2. Choć nie widać bezpośredniego nawiązania do SDLC to każda iteracja zawiera następujące kroki: analiza, zebranie wymagań, implementacja oraz testowanie. Zgodnie z przyjętymi zasadami nie ma miejsca na rozbudowane tworzenie harmonogramu, czy dokumentacji a cały proces skupiony jest na częstym udostępnianiu niewielkich części działającego oprogramowania.

3.4. Miejsce bezpieczeństwa w procesie dostarczania oprogramowania

Niezależnie od tego, jaki model pracy zostanie wybrany do realizacji projektu IT, to procesy związane z dostarczaniem oprogramowania mogą być automatyzowane (proces dostarczenia wytworzonego oprogramowania do klienta końcowego jest wykonywany przez wykorzystywane narzędzia zgodnie ze stworzonymi zasadami bez ingerencji człowieka). Dla modelu kaskadowego proces dostarczania oprogramowania uruchamiany byłby w ostatnim etapie, natomiast dla modelu zwinnego po każdej iteracji. Możliwe jest to dzięki dużej dostępności narzędzi, które służą do: planowania zadań oraz zgłaszania problemów, przechowywania wielu wersji kodu źródłowego, testowaniu aplikacji (typy testów opisane w Tabeli 3.1) oraz budowaniu i uruchamiania oprogramowania w odpowiednim środowisku.

Rysunek 3.3 przedstawia przykładowy łańcuch dostarczania oprogramowania. Programista po otrzymaniu zgłoszenia (na przykład nowa funkcjonalność lub poprawa wykrytego błędu) dostarcza kod źródłowy do repozytorium. Jeśli dostarczony kod spełnia wymagania jakościowe, to na jego podstawie budowana jest nowa wersja aplikacji, która jest



Rys. 3.3: Łańcuch dostarczania oprogramowania

poddawana kolejnym testom po to, aby zweryfikować czy dostarczona przez programistę funkcjonalność działa zgodnie z założeniami. Gdy nowa wersja aplikacji przejdzie poprawnie wszystkie zaplanowane testy, następuje proces jej uruchomienia w odpowiednim środowisku – testowym po to, aby można było wykonać więcej manualnych testów, przed-produkcyjnym, aby zweryfikować zmianę w kontekście całego ekosystemu oraz na końcu produkcyjnym, aby udostępnić wersję klientowi końcowemu.

Zapewnienie bezpieczeństwa dla oprogramowania, które jest tworzone z wykorzystaniem modelu kaskadowego jest zadaniem stosunkowo prostym. Kierownik projektu na etapie planowania oraz tworzenia harmonogramu powinien zarezerwować określony czas, w którym będą przeprowadzone testy bezpieczeństwa. W takim modelu możliwe jest przeprowadzenie testów penetracyjnych systemu informatycznego, które są wykonywane przez wyspecjalizowanego eksperta. Ten typ testów jest najdokładniejszy, ponieważ oprócz weryfikacji tego, czy w stworzonym oprogramowaniu występują podatności bezpieczeństwa, możliwa jest weryfikacja bezpieczeństwa procesów biznesowych, które zostały zaimplementowane.

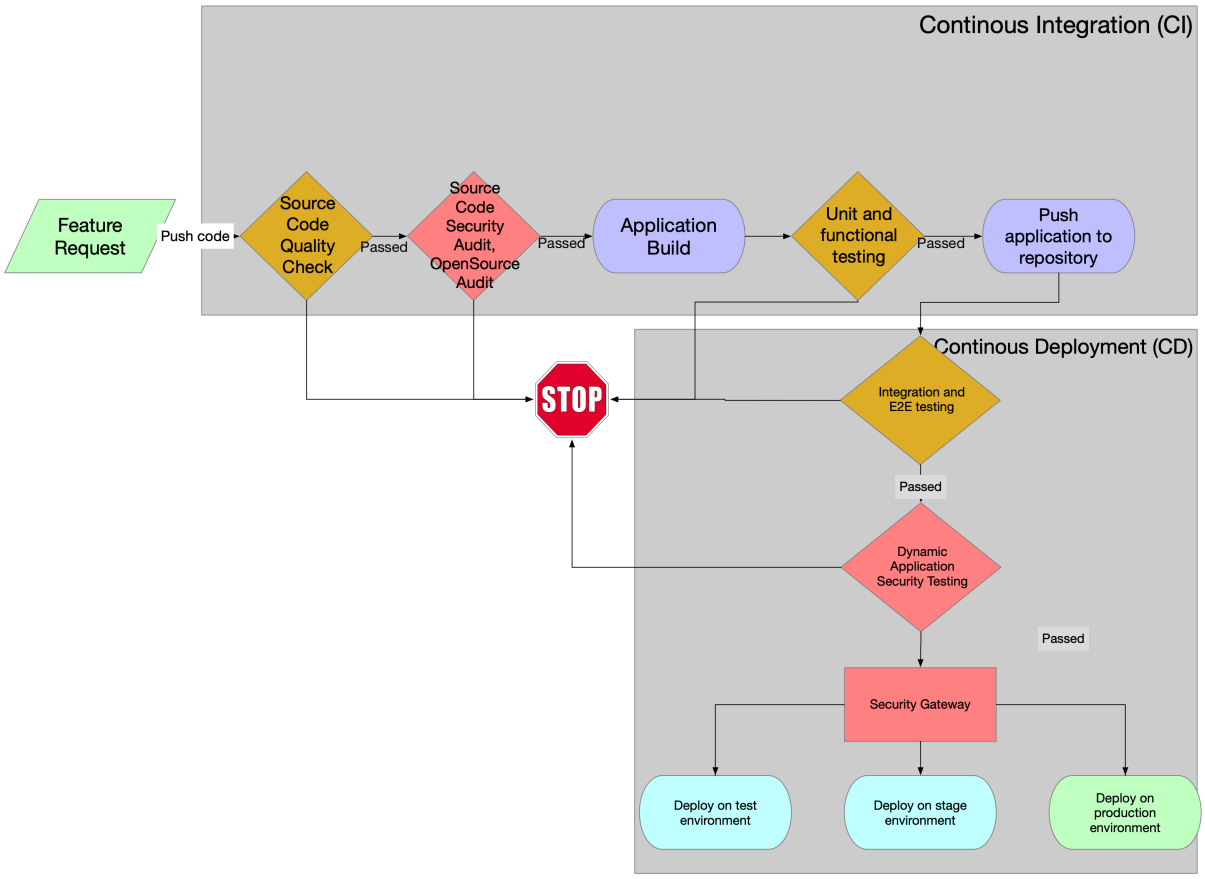
Inaczej realizowane są projekty w modelu zwinnym. Należy pamiętać o tym, że zmiany w kodzie źródłowym (oraz nowe wersje oprogramowania) mogą być udostępniane nawet kilkanaście razy dziennie. Próba umieszczenia w harmonogramie ręcznych testów penetracyjnych (które mogą trwać nawet kilkanaście dni w przypadku dużych projektów) jest sprzeczna z ideą zwinnego zarządzania. Konieczny jest więc kompromis, ponieważ całkowita rezygnacja z mechanizmów weryfikujących bezpieczeństwo może doprowadzić

do materializacji cyberataków takich jak ten z 2017 roku, który doprowadził do wycieku wrażliwych danych 143 milionów obywateli Stanów Zjednoczonych [60]. Mimo braku możliwości realizacji testów penetracyjnych (zbyt długi czas realizacji), możliwe jest wykorzystanie skanerów podatności, czyli narzędzi, które w automatyczny sposób wyszukują luki bezpieczeństwa w zakresie badanego oprogramowania. Testy realizowane przez wspomniane skanery można uruchamiać często, a czas ich realizacji jest akceptowalny z punktu widzenia procesu dostarczania oprogramowania. Istnieje kilka typów skanerów bezpieczeństwa:

- Skanery statyczne [61] - aby wykonać test z wykorzystaniem tego typu skanera wystarczy dostęp do kodu źródłowego rozwiązania. Test polega na weryfikacji linijka po linijce i/lub funkcja po funkcji kodu źródłowego badanego oprogramowania. Skaner typu SAST wyszukuje wzorce, które mogą świadczyć o wystąpieniu podatności w konkretnej linijce kodu. Testy tego typu mogą być często wykonywane Charakteryzują się jednak dużym poziomem fałszywych alarmów, które należy zweryfikować.
- Skanery dynamiczne [62] (ang. Dynamic Application Security Testing - DAST) - aby wykonać test z wykorzystaniem tego typu skanera konieczny jest dostęp do działającej aplikacji, która jest uruchomiona i dostępna przez interfejs WWW. Skany typu DAST polegają na wykonaniu dużej liczby odpowiednio spreparowanych żądań HTTP, które mają na celu złamanie zabezpieczeń rozwiązania (na przykład poprzez wstrzyknięcie kodu JavaScript lub SQL). Dzięki weryfikacji odpowiedzi, która zostanie wysłana przez serwer skaner ocenia, czy badana aplikacja posiada luki bezpieczeństwa.
- Skanery Interaktywne [63] (ang. Interactive Application Security Testing - IAST) - aby wykonać test z wykorzystaniem tego typu skanera konieczny jest dostęp zarówno do kodu źródłowego badanej aplikacji, jak i do uruchomionej aplikacji na dostępnym serwerze. Skan z wykorzystaniem narzędzi IAST weryfikuje testy zarówno statyczne jak i dynamiczne.

Na Rys. 3.4 został przedstawiony zmodyfikowany łańcuch dostarczenia oprogramowania, który zawiera elementy weryfikujące bezpieczeństwo (czerwone bloki). Cały proces wygląda bardzo podobnie do tego, który został zaprezentowany na Rys. 3.3. Dodatkowe bloki odpowiadają za realizację testów SAST oraz DAST po to, aby zestaw testów, dzięki którym podejmowana zostaje decyzja, dotycząca tego czy wprowadzona zmiana jest poprawna, zawierał również badanie zabezpieczeń. Ostatnim dodanym blokiem jest weryfikacja bezpieczeństwa, od której zależy to, czy zmiana, która jest przetwarzana może zostać opublikowana na konkretnym środowisku. Zabieg ten ma uniemożliwić dostarczenie nowej wersji oprogramowania klientowi końcowemu, jeśli wykryto krytyczną podatność bezpieczeństwa.

Opisana powyżej koncepcja, która dodaje odpowiednie mechanizmy weryfikujące bezpieczeństwo opracowywanego oprogramowania w automatyczny sposób jeszcze do



Rys. 3.4: Łańcuch dostarczania oprogramowania zawierający elementy bezpieczeństwa

niedawna była bardzo skomplikowana do zrealizowania z technicznego punktu widzenia. Na rynku rozwiązań teleinformatycznych dostępne są komercyjne aplikacje takie jak ThreadFix[9] czy CodeDX[10], które umożliwiają integracje różnego rodzaju skanerów bezpieczeństwa (SAST oraz DAST), jednak żadne z dostępnych rozwiązań nie implementuje funkcjonalności wspomagającej proces decyzyjny (określenie wpływu konkretnej podatności na konkretny system). Bez tego typu funkcjonalności proces dostarczenia oprogramowania musi zawierać krok związany z ręczną analizą wyników, co znacznie wydłuża czas dostarczenia nowej wersji oprogramowania. Alternatywą do manualnej weryfikacji jest podejmowanie decyzji dotyczącej stanu bezpieczeństwa na podstawie danych zawartych w podatności pobranej ze skanera (na przykład krytyczność), co może spowodować uwzględnienie fałszywych alarmów powodujących przerwanie procesu CICD bez konkretnego powodu. Żadne z obydwu wymienionych podejść nie jest optymalne z perspektywy zapewniania bezpieczeństwa wytwarzanemu oprogramowaniu.

Problemy te adresowane są przez prezentowane w niniejszej rozprawie rozwiązanie. Mixeway umożliwia zarówno integrację z wieloma skanerami typu SAST i DAST oraz zawiera zaimplementowany moduł korelacji wyników, który bazując na algorytmach uczenia maszynowego klasyfikuje podatności na dwie klasy: takie, które są konieczne do poprawy, oraz nieistotne z punktu widzenia polityki bezpieczeństwa. W związku z tym w kolejnym rozdziale zostaną przedstawione algorytmy uczenia maszynowego, które będą wykorzystane do opracowania klasyfikatora podatności bezpieczeństwa.

[9] https://threadfix.it
[10] https://codedx.com

4. Klasyfikacja z wykorzystaniem algorytmów uczenia maszynowego oraz techniki przetwarzania języka naturalnego

Aby rozwiązać problem, komputer potrzebuje algorytmu. Algorytm jest sekwencją instrukcji, które przekształcają daną wejściową w daną wyjściową. Przykładem może być algorytm sortowania, w którym wejściem jest lista liczb natomiast wyjściem ich uporządkowany zbiór. Niestety istnieje wiele zadań, dla których rozwiązania nie ma opracowanego algorytmu - przykładem może być odróżnienie niechcianej korespondencji (SPAM) od tej, która jest poprawna. W tym przypadku wejściem jest wiadomość (lub ciąg znaków, z której ta wiadomość się składa), natomiast wyjściem jest informacja, o tym czy jest ona spamem, czy nie. Nie wiemy jednak jak wykonać transformację danych wejściowych do wyjściowych. Mimo tego posiadamy odpowiednio dużą ilość przykładowych danych, na podstawie których możliwe jest stworzenie wzoru tego, jak wygląda wiadomość, która z dużym prawdopodobieństwem jest spamem [64].

Już w latach 50 XX wieku Artur Samuel określił, że uczenie maszynowe jest to "dziedzina nauki, która daje komputerom umiejętność uczenia bez ich dokładnego programowania" [65]. Temat ten, jak jest to widoczne, jest już badany od bardzo dawna, jednak rozwiązania bazujące na algorytmach uczenia maszynowego zagościły na dobre w życiu codziennym dopiero od kilku lat. Projekty takie jak DeepFaceLab[11] oraz Real-Time-Voice-Cloning[12], są dostępne w serwisie GitHub od początku 2019 roku. Pierwszy z wymienionych, dzięki odpowiednio wytrenowanemu modelowi, pozwala zmienić twarz w dowolnym obrazie na inną wybraną twarz. Natomiast drugi, dzięki udostępnionemu nagraniu potrafi odtworzyć załączony głos dla dowolnego tekstu. Tak duża popularność oraz stopień zaawansowania omawianych projektów spowodowany jest głównie tym, że obecnie mamy dużo większe możliwości, związane z dostępem do ogromnych zestawów danych (zawierających dziesiątki tysięcy próbek) jak na przykład: obrazki, próbki dźwięku czy logi ze zdarzeń sieciowych.

Algorytmy uczenia maszynowego pogrupowane są w zależności od oczekiwanego wyniku. Najpopularniejsze tego typu grupy rozwiązań to: [66]:

- *Uczenie nadzorowane* (ang. Supervised Learning) - najczęściej dotyczy problemów klasyfikacji, gdy dane należy pogrupować w oczekiwaną liczbę klas. O uczeniu z nadzorem mówimy w przypadku, w którym zestaw danych wykorzystywanych do stworzenia funkcji transformującej zawiera informację o tym, do której klasy należy dana próbka.
- *Uczenie bez nadzoru* (ang. Unsupervised Learning) - dotyczy problemów, w których nie mamy dostępu do informacji na temat tego, do której grupy należy próbka.

[11] https://github.com/iperov/DeepFaceLab
[12] https://github.com/CorentinJ/Real-Time-Voice-Cloning

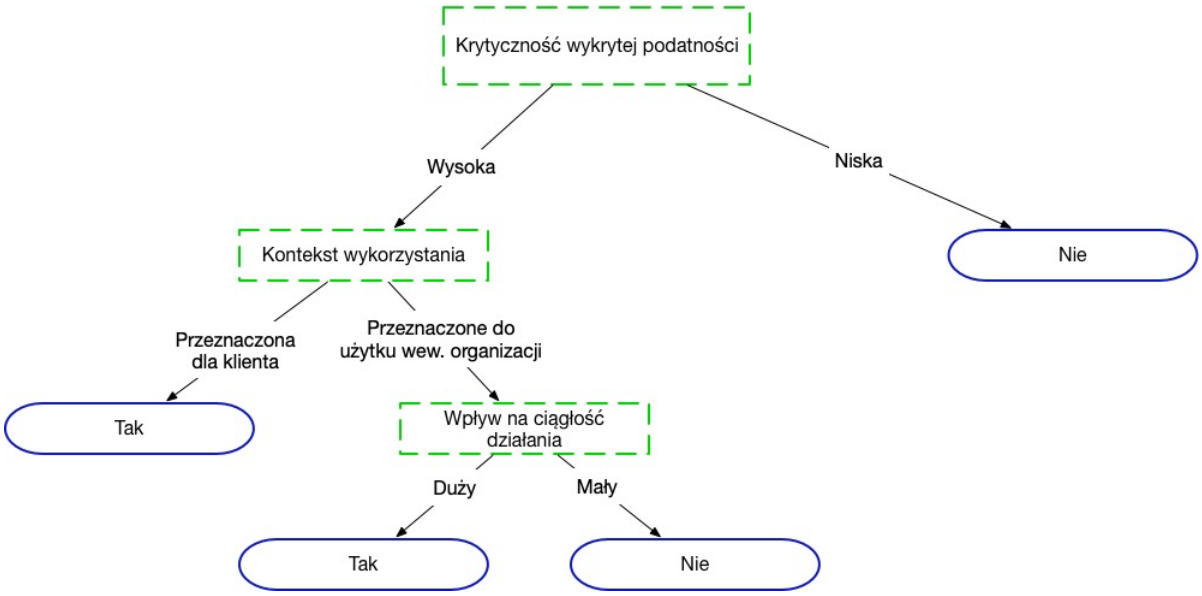
- *Pół-nadzorowane uczenie* - zestaw danych treningowych zawiera zarówno informacje o przynależności do klasy oraz próbki, które tej informacji nie posiadają.
- *Uczenie ze wzmocnieniem* (ang. Reinforcement Learning) - algorytm uczy się polityki, według której ma postrzegać odpowiednie zdarzenia. Każda akcja ma wpływ na otoczenie, które oddziałuje na akcję zwracając odpowiednie informacje algorytmowi.
- *Uczenie wnioskowane* (ang. Transduction) - algorytm w tym przypadku, podobnie jak w uczeniu z nadzorem, posiada informacje o przynależności do klasy próbek treningowych. Celem etapu trenowania nie jest opracowanie funkcji przekształcającej dane wejściowe w wyjściowe, a sposób "przewidzenia" wyniku na podstawie danych wejściowych (danych plus informacji o przynależności).
- *Meta-uczenie* (ang. Learning to learn) - dotyczy sytuacji, w której algorytm uczy się według własnych zasad.

Problemem omawianym w niniejszej rozprawie jest klasyfikacja podatności bezpieczeństwa, które zostały wykryte przez różne źródła (np. skanery podatności lub testy zgodności z wymaganiami). Zestaw danych treningowych, który zostanie podany algorytmowi do nauki zawiera informacje o przynależności do jednej z dwóch klas - CRV (potwierdzona oraz istotna podatność bezpieczeństwa, która musi zostać usunięta) oraz DRNV (wykryta, ale nieistotna podatność). Rozważane będą więc algorytmy uczenia z nadzorem: Klasyfikator Liniowy, Regresja Logiczna, Naiwny Klasyfikator Bayesowski, Sieci Neuronowe, Wektor Maszyny Nośnej, Metoda k-średnich oraz Drzewa i Lasy Losowe [67]). Biorąc pod uwagę charakter danych wejściowych (informacji o podatności), która składa się z danych takich jak: opis podatności, sposób odtworzenia, krytyczność czy lokalizacja istotne będzie wykorzystanie technik przetwarzania języka naturalnego. Pamiętając o wynikach badań prowadzonych przez innych badaczy [41] oraz [42], które udowodniły, że algorytmy takie jak: Sieci Neuronowe, Las Losowy oraz SVM uzyskują najlepsze wyniki w kontekście klasyfikacji podatności bezpieczeństwa tylko te algorytmy będą rozważane oraz porównane w dalszej części pracy. Poniżej, scharakteryzowano zwięźle informacje na temat tego typu rozwiązań.

4.1. Las losowy

Propozycje wykorzystania drzew decyzyjnych w celu rozwiązania problemów klasyfikacji i regresji pojawiały się w literaturze od wczesnych lat 80 XX wieku [68]. Drzewo Decyzyjne jest strukturą, w której węzły wewnętrzne są testami na cechach zbioru danych. Każdy węzeł posiada tyle gałęzi, ile jest możliwych wyników testu w tym węźle. Ostatnim elementem są liście, które zawierają wynik testu. Przykładowe drzewo decyzyjne zostało przedstawione na Rys. 4.1, który zawiera trzy węzły (zielona przerywana linia), cztery liście (niebieska ciągła linia) oraz sześć gałęzi (czarne linie).

Drzewo Decyzyjne jest dobrym wyborem w przypadku zestawu danych o wielu cechach jednak algorytm ten z dużą łatwością uzyskuje stan nadmiernego dopasowania



Rys. 4.1: Przykładowe drzewo decyzyjne dla problemu “Czy podatność wymaga naprawy”

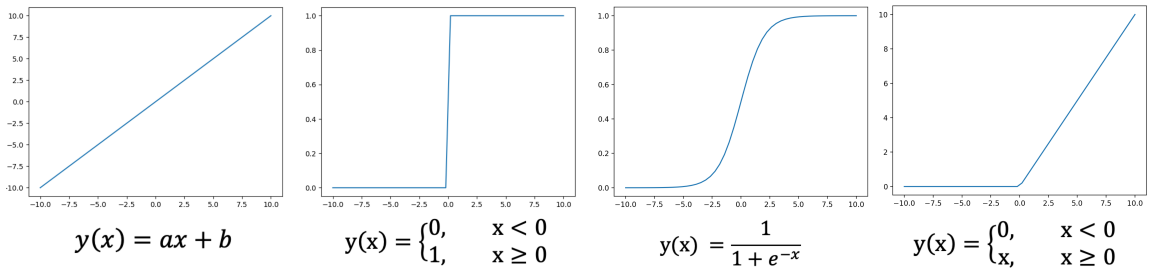
(przeuczenia) do zestawu danych treningowych (ang. overfitting) [69]. Algorytm Lasu Losowego został zaproponowany przez Prof. Breimana w 2001 roku [70] i jest uogólnieniem idei Drzewa Decyzyjnego. Klasyfikacja z wykorzystaniem Lasu Losowego polega na wykorzystaniu określonej liczby wygenerowanych Drzew Decyzyjnych. Decyzja, dotycząca przynależności danej próbki do klasy, podejmowana jest na zasadzie głosowania większościowego przez poszczególne drzewa decyzyjne.

Każde drzewo generowane jest z wykorzystaniem próby bootstrapowej [71] polegającej na losowym wybraniu zarówno próbek z zestawu treningowego jak i cech, które zostaną wzięte pod uwagę. Dzięki takiej operacji uniknięto sytuacji, w której drzewa rozrastają się do bardzo dużych rozmiarów (w sytuacji, gdy zestaw danych treningowych posiada bardzo dużo cech). Przyjmuje się, że liczba cech wykorzystana w konkretnym, stworzonym drzewie jest dużo mniejsza od liczby cech, które znajdują się w zestawie danych treningowych. Zabiegi takie zdecydowanie zmniejszają ryzyko wystąpienia nadmiernego dopasowania do zestawu danych treningowych oraz czynią algorytm Lasu Losowego bardzo skutecznym w rozwiązywaniu problemów klasyfikacji dla zestawów danych, zawierających bardzo dużą liczbę cech.

4.2. Sieci neuronowe

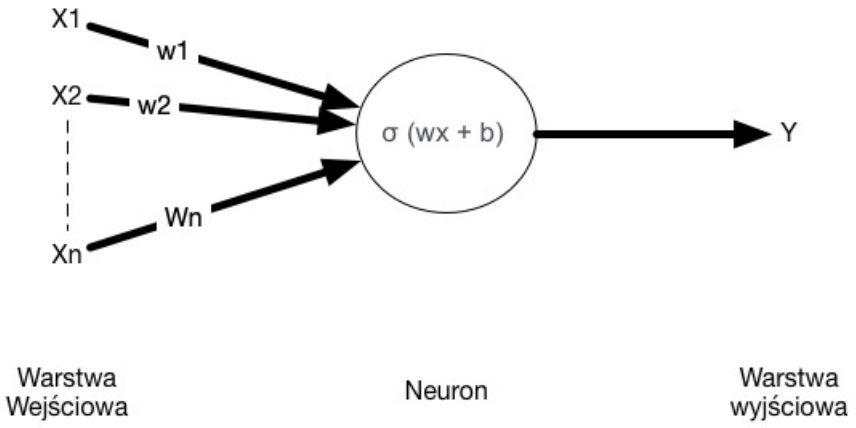
Układ nerwowy człowieka od dawna jest inspiracją dla nauki. Już w 1943 roku przedstawiony został pierwszy model matematyczny neuronu tzw. “Neuron McCullocha-Pittsa” [72]. Zaprezentowany model doprowadził do bardzo dynamicznego rozwoju tej dziedziny nauki oraz był inspiracją do powstania badań nad pierwszym komercyjnym neurokomputerem [73]. Od momentu, gdy wprowadzono pierwsze rodzaje sieci neuronowych, powstało wiele ich typów. Z punktu widzenia problemu omawianego w niniejszej rozpra-

wie najciekawsze typy sieci neuronowych to: sieci rekurencyjne (posiadające sprzężenie zwrotne [74]) oraz sieci konwolucyjne (koncepcja zakładająca wyodrębnianie konkretnych cech [75]).



Rys. 4.2: Najpopularniejsze funkcje aktywacji. Od lewej - funkcja liniowa, binarna funkcja schodkowa, funkcja sigmoidalna, funkcja RELU

Rys. 4.3 przedstawia najbardziej podstawową sieć neuronową, składającą się z jednej jednostki. Etap uczenia sieci polega na dobraniu odpowiednich wag dla każdej cechy (danej wejściowej) $[x_1, x_2, ..., x_n]$. Wagi te wykorzystywane są w celu obliczenia wartości funkcji aktywacji (najbardziej popularne funkcje przedstawione są na Rys. 4.2). Proces nauki modelu zaczyna się wygenerowaniem losowych wartości wag, za pomocą których obliczana jest funkcja błędu (w procesie propagacji sygnału). Następnie wartość funkcji błędu wykorzystywana jest w procesie propagacji wstecznej, aby poprawić wartości wag. Cały proces jest powtarzany dla całego zestawu danych treningowych.



Rys. 4.3: Przykład sieci zbudowanej z jednego neuronu

Wartość predykcji dla pojedynczego przykładu wynosi (zakładając funkcję aktywacji sigmoid, oraz pamiętając, że w oraz x są macierzami zawierającymi wartości dla kolejnych danych wejściowych) $\hat{y} = \sigma(z) = \frac{1}{1-e^{-z}} = \frac{1}{1-e^{-(w.x+b)}}$

Aby rozpocząć proces propagacji wstecznej należy najpierw obliczyć funkcje błędu, która określa jak bardzo wynik predykcji różni się od stanu faktycznego. Również i w tym przypadku możliwe jest dostosowanie postaci funkcji błędu do problemu, który jest roz-

wiązywany. W przypadku problemu regresji wykorzystywany jest błąd średniokwadratowy [76] (ang. Mean Square Error), przy klasyfikacji funkcja kosztu przyjmuje postać średniej entropii krzyżowej [77] (ang. cross entropy):

$$\mathscr{L} = -\frac{1}{N}\sum_{n=1}^{N} t_i(y_n \log z_n + (1-y_n)\log(1-\hat{y}_n)) \tag{1}$$

Aby poprawić wartość wagi (zmienić ją o pewną wartość Δw_i) dla każdej danej wejściowej należy obliczyć wartość funkcji spadku gradientu (ang. gradient descent):

$$\Delta w_i = \frac{\partial \mathscr{L}}{\partial w_i} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i} \tag{2}$$

Obliczając konkretne elementy możemy zaobserwować:

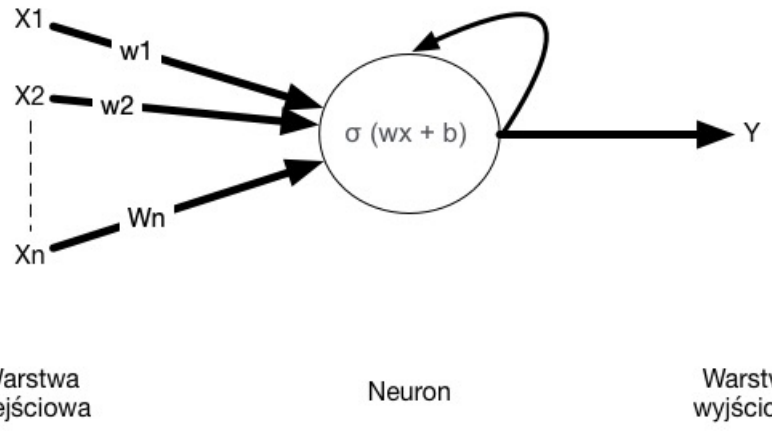
$$\Delta w_i = \frac{\partial \mathscr{L}}{\partial w_i} = \frac{2}{n} \times \sum (y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z)) \tag{3}$$

Zaktualizowana waga, uwzględniając współczynnik nauki (ang. Learing Rate) wynosi:

$$w_i = w_i - (\alpha \times \Delta w_i) \tag{4}$$

Powyższe rozważania zostały wykonane w sposób bardzo uproszczony (m.in. brak biasu, czyli dodatkowego wejścia, na którym występuje stała wartość).

Tak zbudowana Sieć Neuronowa charakteryzuje się jednokierunkowym przepływem sygnału od wejścia do wyjścia. W przypadku sieci rekurencyjnych [74], w których występuje sprzężenie zwrotne (Rys. 4.4) połączenia między neuronami stanowią graf z cyklami (obieg zamknięty). Graf ten tworzy w modelu stan wewnętrzny, przez co głębokość sieci oraz jej pojemność znacznie się zwiększa. Dzięki takiemu zabiegowi sieć zyskuje bardzo duże możliwości nauki. Efektem ubocznym jest znaczne zwiększenie czasu potrzebnego na wytrenowanie modelu. Istnieją również implementacje sieci rekurencyjnych z wyko-



Rys. 4.4: Przykład rekurencyjnej sieci zbudowanej z jednego neurona

rzystaniem jednostek pamięci LSTM (ang. Long Short Term Memory), które zawierają bramki: resetu, aktualizacji i wyjściową. Dzięki swojej budowie sieci te mogą zdecydować, które wartości warto propagować, a które nie. Zapobiega to znikaniu lub ekspozycji błędów propagacji. Dzięki swojej budowie Sieci Neuronowe, wykorzystujące bramki LSTM, osiągają lepsze wyniki dla zestawów danych zawierających bardzo długie przykłady [78]. Pamięć bramkowa [78] (ang. Gated Recurrent Unit, GRU) składa się z jednostek resetu i aktualizacji, przez co posiada mniejszą liczbę parametrów koniecznych do konfiguracji względem LSTM.

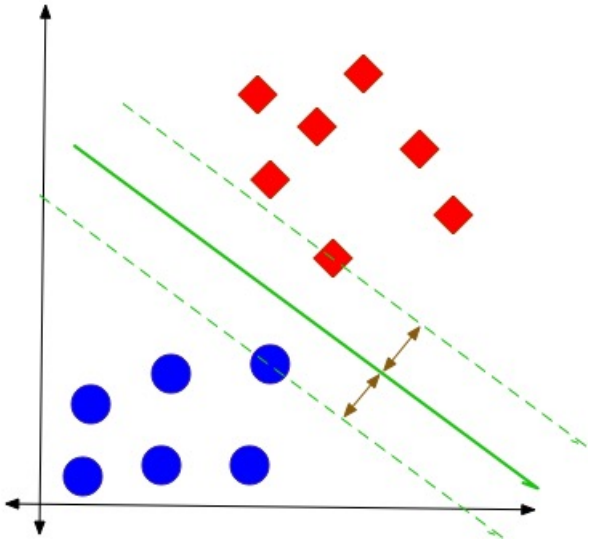
Ostatnim typem Sieci Neuronowej, który zostanie wzięty pod uwagę podczas próby rozwiązania problemu klasyfikacji podatności bezpieczeństwa, jest sieć konwolucyjna. Architektura sieci CNN [75] (ang. Convolutional Neural Network) jest odpowiedzą na problemy ze skalowalnością. Sieć ta składa się z trzech warstw: splotowej, która zbudowana jest z filtrów wyodrębniających konkretne cechy z zestawu danych; głosującej (ang. pooling layer) stosowanej do zmniejszenia wymiaru sieci; łączącej, która wykonuje operacje związane z oceną wyodrębnionych cech w celu nadania konkretnej klasy badanej próbce danych. Sieci CNN powszechnie są wykorzystywane w celu klasyfikacji obrazów, jednak ostatnie badania przedstawiają ich wysoką skuteczność w rozwiązywaniu problemów związanych z klasyfikacją tekstu [79].

4.3. SVM

Algorytm Maszyny Wektorów Nośnych powstał dzięki próbie eliminacji wad sieci neuronowych związanych z charakterem funkcji błędu, który podczas minimalizacji może posiadać wiele minimów lokalnych (które mogą spowodować zatrzymanie procesu uczenia) [80]. Zaproponowane podejście wykorzystuję pomysł polegający na szukaniu maksymalnego marginesu separującego najbliższe punkty w przestrzeni, które definiują oddzielne klasy [81].

Brane pod uwagę są tylko punkty, których oddzielenie jest najtrudniejsze (Rys. 4.5) - odległość między określającymi je wektorami nośnymi (zielona przerywana linia) jest największa. Algorytm SVM tworzy dwuwarstwową strukturę (podobną do Sieci Neuronowej) wykorzystującą różne rodzaje funkcji aktywacji: liniową, wielomianową, radialną lub sigmoidalną. Sposób uczenia natomiast oparty jest na programowaniu kwadratowym, co umożliwia ograniczenie liczby minimów globalnych do jednego [82]. Algorytm SVM wykorzystywany jest najczęściej w celu rozwiązywania problemów klasyfikacji dotyczących dwóch klas oraz zaliczany jest do klasy algorytmów uczenia z nadzorem.

Aby wyjaśnić sposób, który SVM wykorzystuje do znalezienia największego marginesu między skrajnymi punktami wchodzącymi w skład dwóch różnych zbiorów danych, należy



Rys. 4.5: Przykład wyznaczenia maksymalnego marginesu między różnymi zbiorami punktów.

przedstawić reprezentację punktu ϕ, który może być definiowany przez wiele cech:

$$\begin{gathered} x \in \mathbb{R}^D \\ \phi : \mathbb{R}^D -> \mathbb{R}^M, \phi(x) \in \mathbb{R}^M \end{gathered} \tag{5}$$

Granica decyzyjności (zielona ciągła linia na Rys. 4.5) jest $D-1$ wymiarowa i oznacza to, że dla dwuwymiarowej płaszczyzny granica decyzyjności jest prostą, natomiast w przypadku przestrzeni trójwymiarowej mamy do czynienia z płaszczyzną. Możemy ją obliczyć za pomocą równania:

$$\mathscr{H} : w^T \phi(x) + b = 0 \tag{6}$$

Aby policzyć margines separacji (brązowa strzałka na Rys. 4.5), należy znaleźć odległość między badanym punktem, który może być przedstawiony jako (x_0, y_0) od granicy decyzyjności. Wiedząc, że odległość punktu od prostej przedstawionej jako $ax + by + c = 0$ wynosi $\frac{|ax_0+by_0+c|}{\sqrt{a^2+b^2}}$, funkcja marginesu w zależności od punktu wynosi:

$$d_{\mathscr{H}}(\phi(x_0)) = \frac{|w^T \phi(x_0) + b|}{||w||_2} \tag{7}$$

Celem algorytmu SVM jest znalezienie takiej granicy decyzyjności, przy której margines separacji jest największy:

$$w^* = \max_w [\min_n d_{\mathscr{H}}(\phi(x))] \tag{8}$$

Dzięki temu możliwe jest nadanie klasy 1 dla punktów znajdujących się nad hiperpłaszczyzną oraz -1 dla punktów pod nią. Wiedząc, że granica decyzyjności opisywana jest przez równanie 6, możemy określić, że punkty należące do klasy 1 oraz -1 są przedstawione

kolejno przez: $w^T\phi(x)+b>0$ oraz $w^T\phi(x)+b<0$ co można uprościć jako:

$$y_n[w^T\phi(x)+b] = \begin{cases} \geq 0, & \text{klasa 1} \\ < 0, & \text{klasa 2} \end{cases} \tag{9}$$

Problem optymalizacyjny jest przedstawiany jako:

$$w^* = \max_w \frac{1}{||w||_2}[\min_n y_n[w^T\phi(x)+b]] \tag{10}$$

Przy założeniu idealnej separacji punktów należących do różnych klas: $\min_n y_n[w^T\phi(x)+b]=1$ podstawowa forma algorytmu SVM to:

$$\min_w \frac{1}{2}||w||_2 \tag{11}$$

Niestety podczas rozwiązywania problemów rzadko spotykamy się z zestawami danych, które charakteryzują się tym, że dane treningowe należące do konkretnych klas są widocznie od siebie odseparowane. W przypadku klasyfikatora o nieliniowych granicach decyzyjności stosuje się metodę mnożników Lagrange'a [83], postać dualną lub dostosowanie funkcji jądrowej (ang. Kernelization) [84].

4.4. Przetwarzanie języka naturalnego

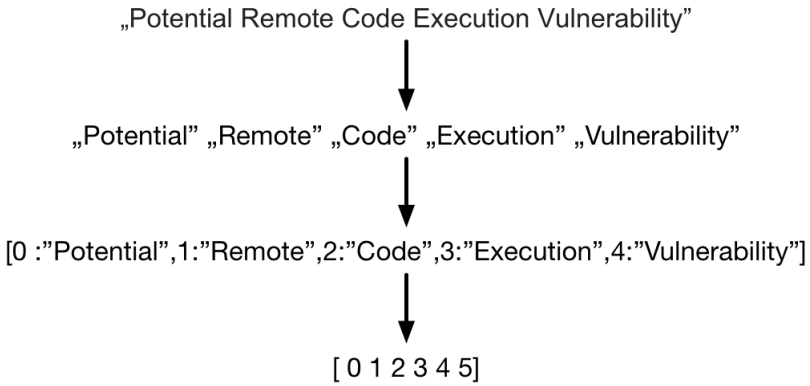
Przetwarzanie języka naturalnego - NLP (ang. Natural Language Processing) jest dziedziną łączącą tematy związane z lingwistyką, informatyką oraz uczeniem maszynowym. NLP skupia się na badaniu interakcji pomiędzy komputerami a ludzkim językiem, a w szczególności tym jak zaprogramować komputery do przetwarzania i analizowania dużych ilości danych w języku naturalnym. Ta dziedzina nauki ma swoje początki w latach 50 dwudziestego wieku, po zaprezentowaniu przez Alana Turinga "Testu Turinga"[85], który określa zdolności oprogramowania do rozumienia tekstu napisanego z wykorzystaniem języka naturalnego.

Prace badaczy, opisujące to zagadnienie w ostatnich latach prezentują nowe narzędzia wspomagające rozwiązywanie problemów [86] oraz udowadniają, że możliwe jest korzystanie z technik NLP w połączeniu z różnymi algorytmami ML takimi jak Sieci Neuronowe [87]. Do podstawowych zastosowań systemów NLP można zaliczyć:

- Rozpoznawanie i syntezę mowy [88].
- Prowadzenie dialogu maszyna-człowiek [89].
- Analizę nastroju tekstu (ang. Sentiment Analysis) [90].
- Rozumienie i generowanie tekstu naturalnego.
- Automatyczne tłumaczenie tekstu (oraz korekcja błędów językowych) [91].

Jednym z najważniejszych etapów podczas korzystania z technik NLP jest przetwarzanie wstępne badanego tekstu. Zadanie to pozwala na przygotowanie zestawu danych

tak, aby możliwe było stworzenie modelu, który pomoże rozwiązać zadany problem [92]. W zależności od zestawu danych przetwarzanie wstępne może zostać wykonane przez tokenizację, która jest kilkuetapową operacją. Pierwszy krok wspomnianego procesu polega na wyodrębnieniu wyrazów z tekstu. W tym etapie budowany jest słownik wszystkich leksemów (wyrazów słownikowych), które są wykorzystywane w opisie podatności. Następnie każde słowo zamieniane jest na pozycje (indeks) w słowniku. Drugą techniką jest segmentacja, która wymaga oznaczenia granic wyrazów (szczególnie istotne przy językach takich jak: japoński i tajski, w których nie ma określonych granic wyrazów). Innym sposobem jest oznaczanie części mowy - POS (ang. Part of Speech), które polega na określeniu części mowy dla każdego słowa. Zabieg ten pozwala na wyodrębnienie takiego samego słowa, które w zależności od kontekstu może być czasownikiem lub rzeczownikiem.

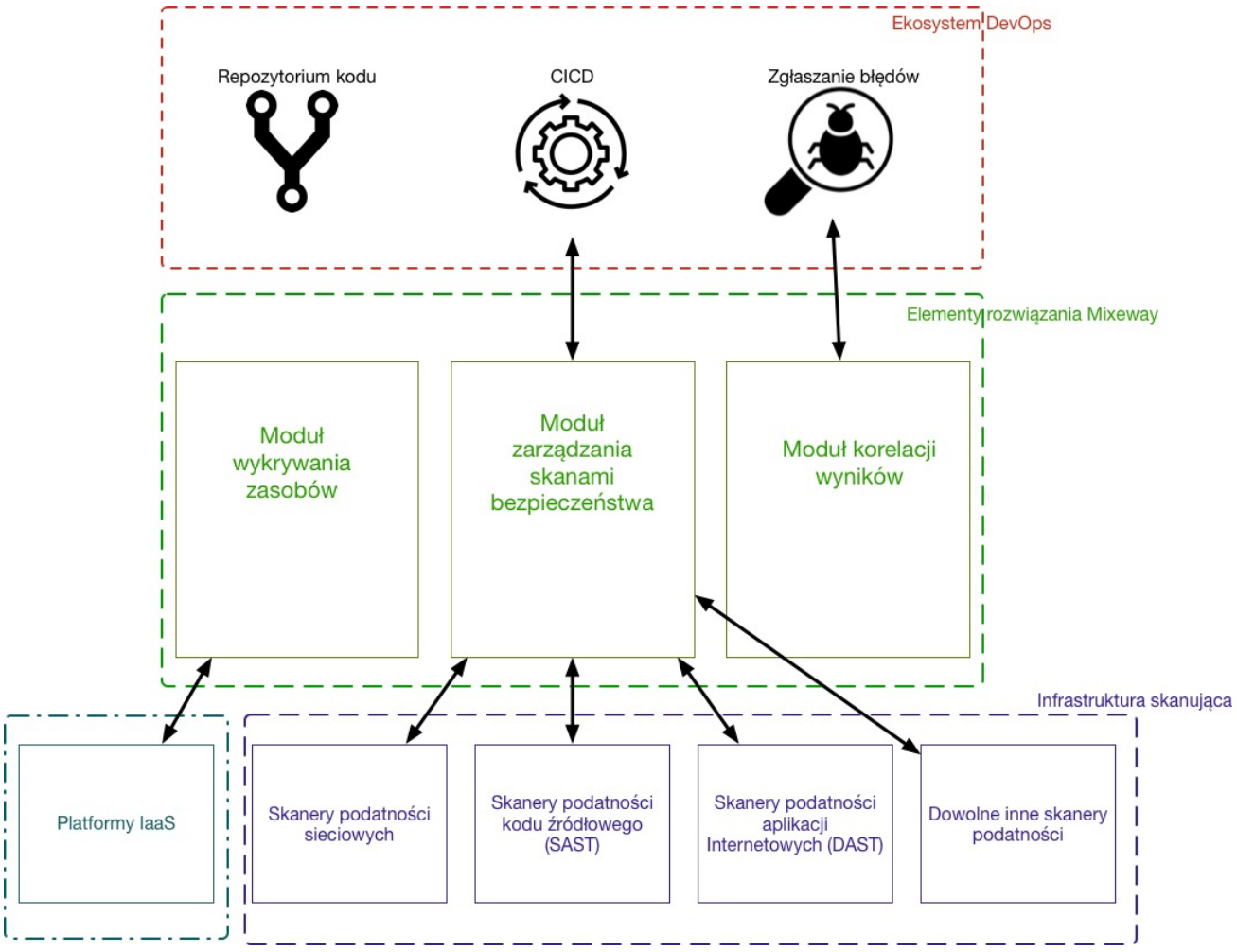


Rys. 4.6: Sposób działania tokenizacji na przykładzie pola zawierającego nazwę podatności

W rozwiązaniu opisywanym w niniejszej rozprawie źródłem danych, które zasilą opracowywany klasyfikator są skanery podatności. Biorąc pod uwagę informację, jakie znajdują się w konkretnych polach opisujących wykryte podatności bezpieczeństwa (np. nazwa podatności, opis, sposób odtworzenia czy propozycja naprawy), zastosowanie będą miały techniki z analizą nastroju tekstu - weryfikowanie czy dana próbka wymaga poprawy (“CRV”) lub jest nieistotna (“DRNV”). Technika, która zostanie wykorzystana w eksperymencie opisanym w rozdziale 6 to tokenizacja. W kolejnych rozdziałach zostanie przedstawiony system Mixeway, w ramach którego został zaimplementowany opisany klasyfikator oraz wyniki eksperymentu polegającym na porównaniu wyników uzyskiwanych prze modele opracowane z wykorzystaniem różnych algorytmów.

5. Mixeway: architektura i sposób wdrożenia

W celu realizacji zadań, umożliwiających dodanie mechanizmów bezpieczeństwa do procesu CICD zaimplementowane zostało rozwiązanie Mixeway. Rys. 5.1 przedstawia wysokopoziomową koncepcję prezentowanego rozwiązania. Oprogramowanie składa się z trzech głównych modułów: *wykrywania zasobów* (przedstawiony w rozdziale 5.1.1), *zarządzania skanami bezpieczeństwa* (rozdział 5.1.2) oraz *korelacji wyników* (rozdział 5.1.3). Całość wytworzonego w ramach rozprawy doktorskiej oprogramowania jest oparta na otwartym kodzie źródłowym i jest udostępniona publicznie za pośrednictwem serwisu GitHub[13] (kod źródłowy) oraz DockerHub[14] skąd można pobrać obrazy konkretnych elementów systemu. W momencie tworzenia niniejszej rozprawy aplikacja została pobrana ponad 14 000 razy.



Rys. 5.1: Elementy realizujące łańcuch dostarczania oprogramowania przedstawione na Rys. 3.4 z wykorzystaniem prezentowanego rozwiązania

[13] https://github.com/Mixeway/MixewayHub
[14] https://hub.docker.com/u/mixeway

Koncepcja omawianego systemu integruje kilka obszarów (Rys. 5.1). Pierwszym z nich jest ekosystem narzędzi wykorzystywanych przez zespoły DevOps, na który składa się między innymi repozytorium kodu źródłowego, CICD oraz systemy zgłaszania błędów (w szczególności może to być jedno rozwiązanie, które realizuje wszystkie 3 zadania). Dzięki temu możliwe jest skonfigurowanie łańcucha dostarczania oprogramowania tak, aby Mixeway był informowany każdorazowo, gdy budowana jest nowa wersja aplikacji (informacja o nazwie, adresie repozytorium oraz gałęzi kodu). Za każdym razem, gdy nowa podatność bezpieczeństwa zostaje wykryta zespoły programistyczne są informowane o tym fakcie za pośrednictwem systemu zgłaszania błędów. Dane pozyskiwane z tego rodzaju informacji pozwalają na odpowiednie skonfigurowanie testów zarówno SAST (skanowane jest repozytorium kodu, w którym znajduję się aplikacja) oraz DAST (skanowany jest konkretny URL).

Drugim obszarem, który zapewnia dodatkowe dane umożliwiające precyzyjne zdefiniowane zakresu testów podatności są platformy IaaS (np. Google Cloud Platform). Za pomocą tej integracji Mixeway w czasie zbliżonym do czasu rzeczywistego pozyskuje informacje na temat uruchomionych zasobów (nazwa maszyny wirtualnej oraz jej adres IP). Dzięki dostępności tych danych możliwe jest odpowiednie skonfigurowanie skanów sieciowych.

Kolejnym elementem jest infrastruktura testująca, na którą składają się skanery: sieciowe, SAST oraz DAST. Dane pozyskane zarówno z ekosystemu DevOps oraz z platform IaaS zasilają moduł zarządzania podatnościami, dzięki czemu proces konfigurowania oraz uruchamiania testów podatności jest automatyczny jak i bezobsługowy.

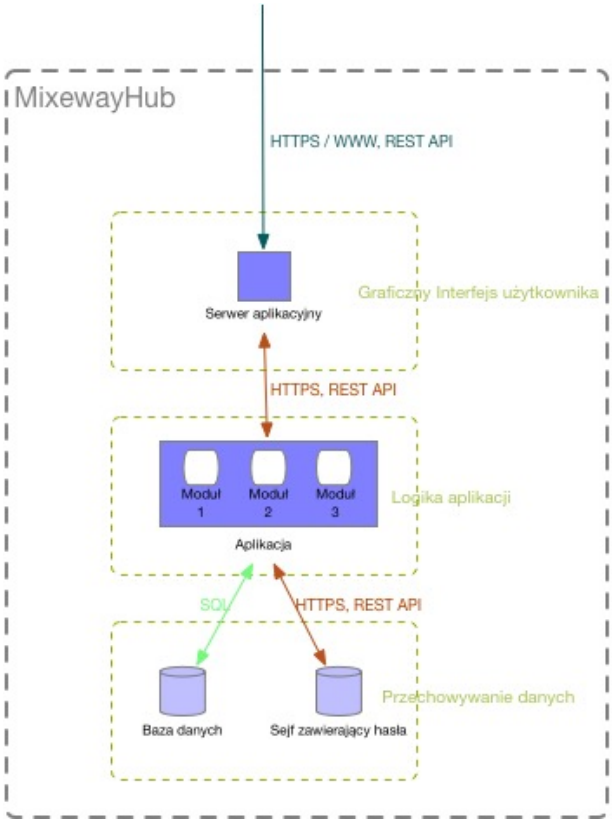
5.1. Architektura

Niskopoziomową architekturę stworzonego oprogramowania – Mixeway – przedstawiono na Rys. 5.2.

Całość rozwiązania została podzielona na trzy logiczne elementy. Pierwszym z nich jest graficzny interfejs użytkownika, który składa się z aplikacji zbudowanej z wykorzystaniem technologii Angular[15] [93] oraz serwera aplikacyjnego Nginx, za którego pośrednictwem aplikacja jest udostępniana dla użytkowników. Szata graficzna bazuje na rozwiązaniu "Ngx-admin"[16]. To jeden z obecnie najbardziej popularnych szablonów do tworzenia aplikacji internetowych o otwartym kodzie źródłowym.

Drugi element jest odpowiedzialny za realizację logiki biznesowej rozwiązania. Komponent ten został zaimplementowany z wykorzystaniem technologii Spring Boot (Java). Świadomie zrezygnowano z architektury mikroserwisowej [94], która polega na jak największej dekompozycji rozwiązania na usługi realizujące konkretne zadania. Podejście takie jest szczególnie wskazane przy bardzo rozbudowanych rozwiązaniach, w których

[15] https://angular.io
[16] https://github.com/akveo/ngx-admin



Rys. 5.2: Niskopoziomowa architektura systemu Mixeway

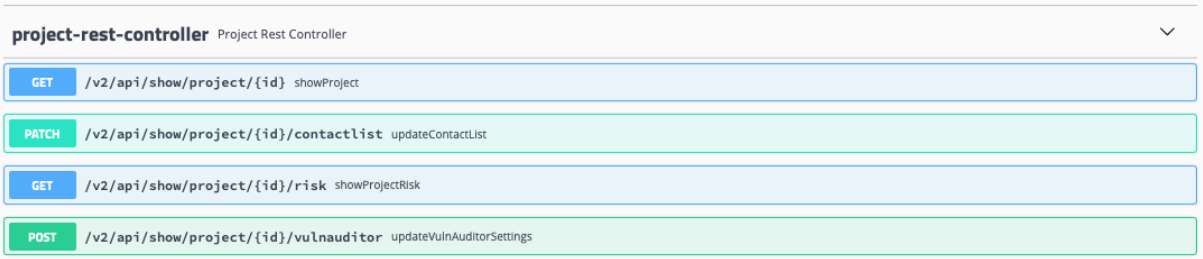
istotny jest czas dostarczania nowych wersji oprogramowania użytkownikom końcowym. Dzięki podzieleniu aplikacji na dużo mniejsze usługi możliwe jest aktualizowanie konkretnego elementu systemu bez ryzyka utraty dostępności całego rozwiązania. W przypadku oprogramowania Mixeway udostępnianego jako aplikacja, którą instaluje się na infrastrukturze zainteresowanego użytkownika zabieg podziału na mikroserwisy nie wprowadza tak wielu korzyści. Zamiast tego aplikacja została podzielona na moduły, z których każdy budowany jest z wykorzystaniem wtyczki do konkretnych zastosowań tj. na podstawie konkretnych szablonów, co znacznie ułatwia utrzymanie aplikacji oraz umożliwia innym programistom tworzenie swoich rozszerzeń. Szczegółowy opis opracowanych modułów został przedstawiony w rozdziałach 5.1.1- 5.1.3.

Ostatnim elementem architektury niskopoziomowej Mixeway jest przechowywanie danych, na które składa się baza danych - PostgreSQL[17] oraz sejf, który przechowuje hasła do różnego rodzaju zasobów - Hashicorp Vault[18]. W kontekście omawianego rozwiązania, w którym wolumen przechowywanych danych nie jest zbyt duży (na przykład w porównaniu do systemów przechowujących informacje na temat ruchu sieciowego), a relacje między konkretnymi rekordami są istotne, wybrany został jeden z najbardziej popularnych, darmowych silników bazodanowych [95]. Przechowywanie haseł w przypadku

[17] https://www.postgresql.org
[18] https://www.vaultproject.io

operacji logowania użytkownika jest stosunkowo prostym zadaniem. Hasło podane przez użytkownika należy poddać operacji wyliczenia funkcji skrótu (stosując odpowiednio przygotowaną “sól” i “pieprz”) [96], a następnie tak przygotowany ciąg znaków można bezpiecznie przechować w bazie danych. Operacja weryfikowania hasła podanego przez użytkownika realizowana jest na podobnej zasadzie - hasło wpisane w formularzu poddawane jest podobnym zabiegom po to, aby na końcu porównać, czy wynik działania funkcji skrótu jest podobny. Hasło w postaci jawnej nigdy nie powinno być przechowywane w formie, która umożliwia dostęp do jego oryginalnej postaci. W przypadku rozwiązania Mixeway, które loguje się z wykorzystaniem wcześniej podanych danych uwierzytelniających do konkretnych systemów (na przykład skanerów podatności), wcześniej opisana operacja będzie nieskuteczna. Mixeway musi mieć dostęp do jawnej formy hasła po to, aby wykorzystać je podczas nawiązywania połączenia z danym zasobem (takim jak na przykład oprogramowanie wykonujące test podatności). Aby zachować odpowiedni poziom bezpieczeństwa przechowywanych danych, przy tak zdefiniowanych wymaganiach, wykorzystano rozwiązanie HashiCorp Vault [97]. Dzięki odpowiednio przygotowanym interfejsom programistycznym Vault potrafi zaszyfrować dowolny ciąg znaków z wykorzystaniem bezpiecznych algorytmów, a następnie tę operację odwrócić, aby udostępnić hasło w jawnej formie. Dzięki temu przechowywane hasło jest zaszyfrowane a możliwość jego odszyfrowania ma jedynie Mixeway.

Całość została zaprojektowana zgodnie z podejściem “API First” [98], które zakłada umożliwienie wykonania dowolnej operacji w kontekście systemu za pośrednictwem interfejsu programistycznego. Tak zbudowaną aplikację można w dokładny sposób udokumentować z wykorzystaniem standardu OpenAPI na przykład z wykorzystaniem formatu Swagger[19] [99], przedstawiającego każdą dostępną metodę wraz z definicją danych, które są potrzebne do zrealizowania konkretnej operacji. Przykład definicji konkretnego kontrolera przedstawiono na Rys. 5.3.



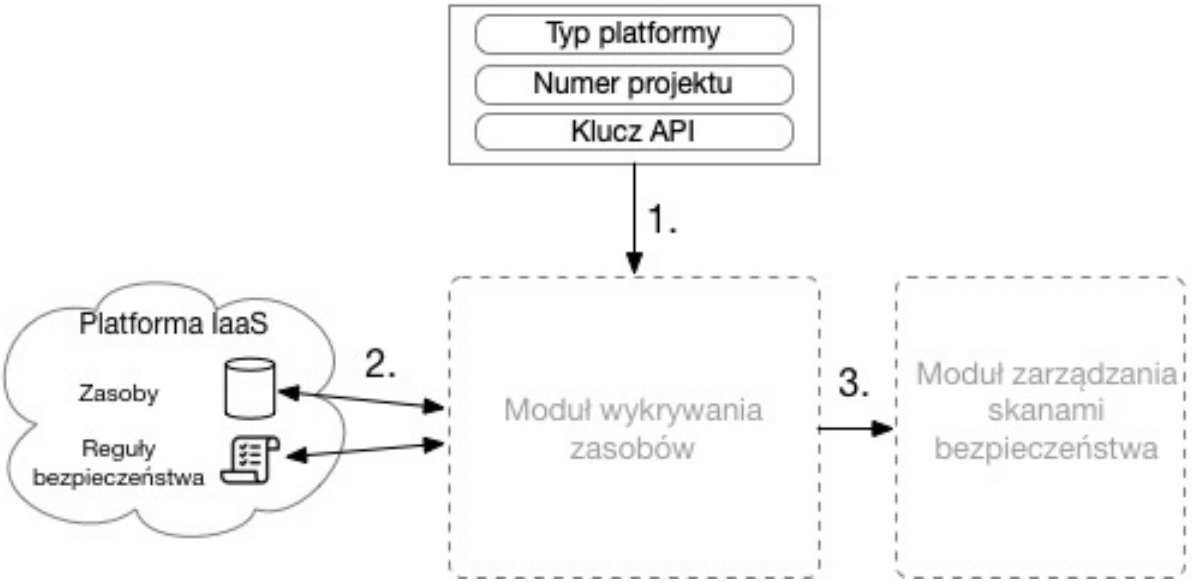
Rys. 5.3: Definicja API wykonującego operacje na danym projekcie wewnątrz Mixeway

5.1.1. Wykrywanie zasobów

Moduł wykrywania zasobów służy do integracji rozwiązania Mixeway z różnego rodzaju platformami, na których dostarczana jest infrastruktura. Może być to zarówno

[19] https://swagger.io

lokalna infrastruktura, która jest wirtualizowana z wykorzystaniem platformy OpenStack[20] lub rozwiązania chmury publicznej takie jak Google Cloud Platform[21] czy Amazon WebServices[22]. Aby zainicjować moduł należy w pierwszej kolejności dostarczyć dane dotyczące



Rys. 5.4: Sposób działania modułu wykrywania zasobów

przestrzeni na platformie IaaS (ang. Infrastructure as a Service), której dotyczy integracja (np. indywidualny numer projektu) oraz klucz API służący do uwierzytelnienia (krok 1). Następnie z wykorzystaniem podanych wcześniej danych moduł wykrywania zasobów nawiązuje połączenie z platformą po to, aby pobrać informacje dotyczące zasobów np. maszyn wirtualnych (krok 2), które są uruchomione w danym kontekście. Zakres pobieranych danych zawiera informacje na temat: adresacji IP, nazwy zasobu oraz sieci, w której znajdują się interfejsy maszyny wirtualnej. Drugim typem informacji, pobieranej z platformy IaaS są definicje grup bezpieczeństwa, do których przynależy dany zasób. Reguły te zawierają informacje opisujące jakiego typu komunikacja jest możliwa (definicja reguł dla ruchu sieciowego przychodzącego i wychodzącego). Wszystkie te dane są wykorzystywane do zdefiniowania oraz zrealizowania testów bezpieczeństwa.

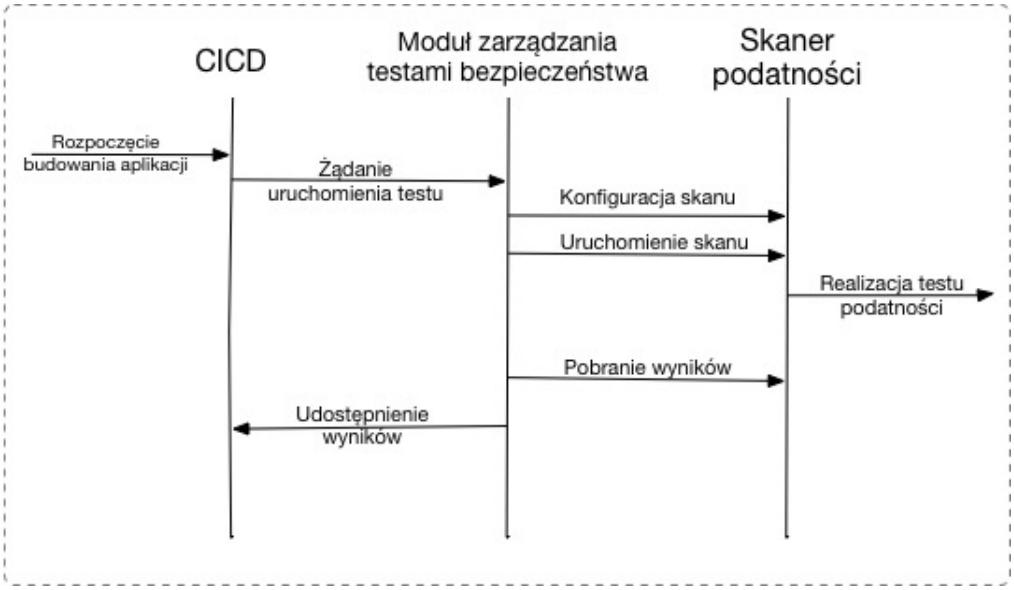
Dzięki operacjom wykonywanym przez moduł wykrywania zasobów zakres skanów podatności w warstwie sieciowej jest definiowany automatycznie bez ingerencji człowieka. Za każdym razem, gdy zostanie wykryta nowa maszyna wirtualna uruchomiona na platformie, zostaje ona dodana do zakresu testów podatności. Operacje wykonywane w taki sposób pozwalają na reakcje w czasie zbliżonym do czasu rzeczywistego za każdym razem, gdy wykryty zostanie zasób podatny na konkretny typ ataku informatycznego, na przykład przez uruchomioną usługę zawierającą znane podatności bezpieczeństwa

[20] https://www.openstack.org
[21] https://cloud.google.com
[22] https://aws.amazon.com

lub udostępnienie interfejsu SSH z domyślnymi danymi uwierzytelniającymi takimi jak admin/admin.

5.1.2. Zarządzanie skanami bezpieczeństwa

Obecnie większość skanerów podatności posiada odpowiednio przygotowane wtyczki do rozwiązań CICD, które pozwalają na realizację testów bezpieczeństwa bezpośrednio podczas wykonywania zadań związanych z budowaniem aplikacji. Jednakże warto zauważyć, że znaczna liczba dużych organizacji korzysta z co najmniej kilku rozwiązań służących do wykonywania automatycznych testów podatności. W takiej sytuacji posiadanie modułu zarządzania testami bezpieczeństwa (realizującego komunikację ze skanerami) znacznie ułatwia utrzymanie procesu zarządzania podatnościami bezpieczeństwa w organizacji.

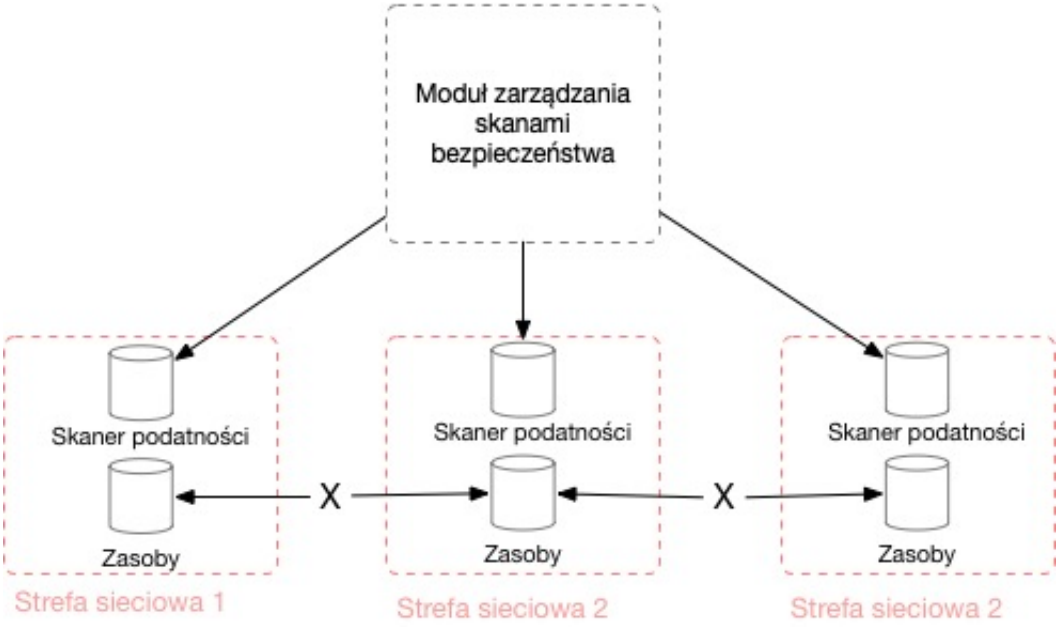


Rys. 5.5: Sposób działania modułu zarządzania skanami bezpieczeństwa

Rys. 5.5 przedstawia operacje, jakie wykonywane są od momentu zlecenia testu do jego wykonania. Operacje według tego schematu wykonywane są również przez inne aplikacje takie jak ThreadFix [100] lub CodeDX [101]. Niestety wspomniane rozwiązania nie są przydatne w sytuacji, w której organizacja podzielona jest na strefy sieciowe [102] w celu separacji na przykład systemów telekomunikacyjnych od systemów sprzedażowych w taki sposób, aby nie była możliwa komunikacja między nimi.

Rys. 5.6 przedstawia sytuację, w której w infrastrukturze firmy znajdują się trzy strefy sieciowe nieosiągalne dla siebie nawzajem. W takiej sytuacji, aby mieć możliwość wykonywania testów podatności zasobów, konieczne jest posiadanie skanerów podatności w każdej ze stref tak, aby badane zasoby były dostępne dla skanera. Moduł zarządzania skanami bezpieczeństwa posiada umiejętność inteligentnej detekcji, która pozwala na wykrycie odpowiedniego skanera, który jest właściwy do uruchomienia testu.

Obecna wersja Mixeway (v1.4.1) posiada wtyczki pozwalające na integrację z:



Rys. 5.6: Sposób działania modułu zarządzania skanami bezpieczeństwa

- Skanery statyczne:
  - Microfocus Fortify[23],
  - Checkmarx[24].
- Skanery dynamiczne:
  - Burp Enterprise Edition [25],
  - Acunetix[26].
- Skanery sieciowe:
  - Nessus Proffesional (wersje < 7) [27],
  - Nexpose[28],
  - Greenbone Vulnerability Manager[29] (dawniej OpenVAS).
- Skanery bibliotek:
  - OWASP Dependency Track[30].

Moduł zarządzania skanami został zaprojektowany w sposób, który pozwala na modularne dodawanie integracji z kolejnymi skanerami podatności. Aby stworzyć nową wtyczkę, która umożliwia na korzystanie z nowego skanera podatności (nieznajdującego

[23] https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer
[24] https://www.checkmarx.com
[25] https://portswigger.net/burp/enterprise
[26] https://www.acunetix.com
[27] https://www.tenable.com/products/nessus/nessus-professional
[28] https://www.rapid7.com/products/nexpose/
[29] https://www.greenbone.net/vulnerability-management/
[30] http://dependencytrack.org

się na liście powyżej), wystarczy stworzyć klasę implementującą interfejs. Przykład dla integracji ze skanerem typu SAST znajduję się na Rys. 5.7.

```
/**
 * Interface for plugins that integrates with SAST scanners
 */
public interface CodeScanClient {
    /**
     * Function which allows to load vulnerabilities for given application from remote scanner
     */
    void loadVulnerabilities(Scanner scanner, CodeGroup codeGroup, String urlToGetNext, Boolean single, CodeProject codeProject,
    /**
     * Function which run particular scan on remote scanner. Boolean result of operation is returned.
     */
    Boolean runScan(CodeGroup cg,CodeProject codeProject) throws CertificateException, UnrecoverableKeyException, NoSuchAlgorith
    /**
     * Function which check if state of particular scan is done.
     */
    boolean isScanDone(CodeGroup cg, CodeProject cp) throws CertificateException, UnrecoverableKeyException, NoSuchAlgorithmExce
    /**
     * Function which verify if particular scanner is authorized to perform scan on given application.
     */
    boolean canProcessRequest(CodeGroup cg);
    /**
     * Function which verify if particular scanner is authorized to perform scan.
     */
    boolean canProcessRequest(Scanner scanner);
    /**
     * Function which loads already defined projects from remote scanner.
     */
    List<SASTProject> getProjects(Scanner scanner) throws CertificateException, UnrecoverableKeyException, NoSuchAlgorithmExcept
    /**
     * Function which create project entity on remote scanner.
     */
    boolean createProject(Scanner scanner, CodeProject codeProject) throws CertificateException, UnrecoverableKeyException, NoSu
}
```

Rys. 5.7: Interfejs zawierający metody pozwalające na integrację ze skanerami podatności typu SAST

5.1.3. Korelacja wyników

Raporty generowane przez narzędzia służące do wykonywania testów podatności często zawierają setki stron. Odbiór tego typu dokumentów jest trudny zarówno dla osób odpowiedzialnych za bezpieczeństwo (zgłaszający), jak i adresatów, którymi są administratorzy infrastruktury lub programiści. Należy zwrócić uwagę, że raporty generowane przez skanery działające w różnych warstwach systemu wskazują na podatności, które można interpretować w różny sposób. Nie istnieje więc jedna uniwersalna metoda pozwalająca na wytypowanie najważniejszych podatności z otrzymanego raportu. Stwierdzenie, że tylko podatności o krytyczności "Wysoka" muszą być zgłaszane do odpowiednich zespołów, a następnie poprawiane nie jest odpowiednie, ponieważ:

- Podatności o krytyczności "Średnia" lub "Niska", które są zgłaszane przez narzędzia, mogą łamać zdefiniowaną w firmie politykę bezpieczeństwa.
- Narzędzia typu SAST charakteryzują się wysokim procentem wykryć fałszywie pozytywnych.

- Decyzja o tym, czy dana podatność musi zostać usunięta zależy od kontekstu, w jakim została wykryta. Innym zasadom podlegają systemy działające produkcyjnie (dostępne dla klientów końcowych) a innym systemy testowe.

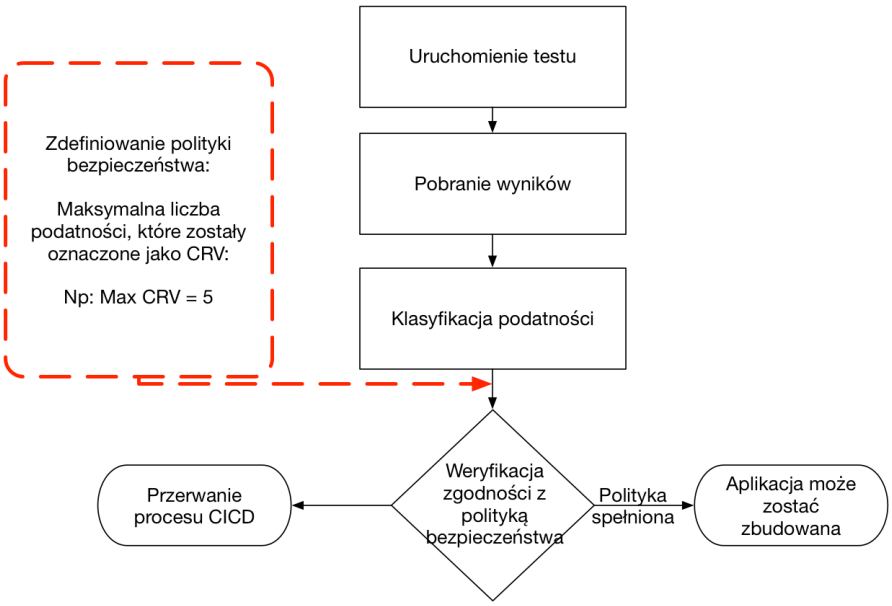
Aby możliwa była szybka decyzja dotycząca tego, które ze zgłoszonych błędów są konieczne do poprawy w oprogramowaniu Mixeway wprowadzono dodatkową metrykę - "CRV" (dla podatności koniecznych do poprawienia) oraz "DRNV" (dla nieistotnych błędów).

Moduł korelacji wyników bazuje na wynikach eksperymentu opisanego w rozdziale 6, na podstawie którego powstał model Sieci Neuronowej. Wspomniany model bierze pod uwagę informacje takie jak: miejsce wystąpienia podatności, opis i nazwę błędu oraz kontekst, w jakim działa badana aplikacja. Dodatkowo określa czy defekt musi zostać zgłoszony przez systemy typu zgłaszania błędów (a następnie usunięty). W ten sposób zespoły odpowiedzialne za działanie aplikacji otrzymują informację dotyczącą konkretnych naruszeń w strukturyzowany sposób.

Instalacja oraz sposoby wykorzystania modelu korelacji wyników zostały szczegółowo opisane w Załączniku 1. Kod źródłowy opublikowany jest w repozytorium GitHub[31]. Aplikację tę można uruchomić na dwa sposoby. Pierwszym z nich jest jej uruchomienie z wykorzystaniem wirtualnego środowiska Python. Korzystając z tego rozwiązania, oprócz możliwości uruchomienia API, które realizuje logikę opracowanego klasyfikatora, istnieje możliwość odtworzenia eksperymentu na dowolnym zestawie danych zawierającym listę podatności bezpieczeństwa w odpowiednim formacie. Drugi sposób, który jest zalecany podczas instalacji oprogramowania Mixeway to skorzystanie z przygotowanego obrazu docker'owego [103][32]. Skorzystanie z drugiego wariantu pozwala pominąć proces nauki modelu, ponieważ udostępniony obraz zawiera w sobie implementację modelu, którego wyniki omawiane są w niniejszej rozprawie.

Dodatkowym elementem omawianego modułu jest bramka bezpieczeństwa, która bazując na wyniku klasyfikacji udostępnia informację dotyczące tego, czy weryfikowana aplikacja spełnia przyjętą politykę bezpieczeństwa. Sposób działania bramki bezpieczeństwa został przedstawiony na Rys. 5.8. Pierwszym krokiem jest zdefiniowanie polityki bezpieczeństwa tak, aby uwzględniała ona klasyfikację realizowaną przez moduł korelacji wyników. Krok ten polega na określeniu maksymalnej liczby podatności bezpieczeństwa, które zostały wykryte w badanej aplikacji oznaczonych jako CRV. Przykładowo zdefiniowanie $maxCRV = 5$ powoduję zatrzymanie łańcucha CICD dla każdej aplikacji, w której wykryto więcej niż 5 podatności sklasyfikowanych jako CRV.

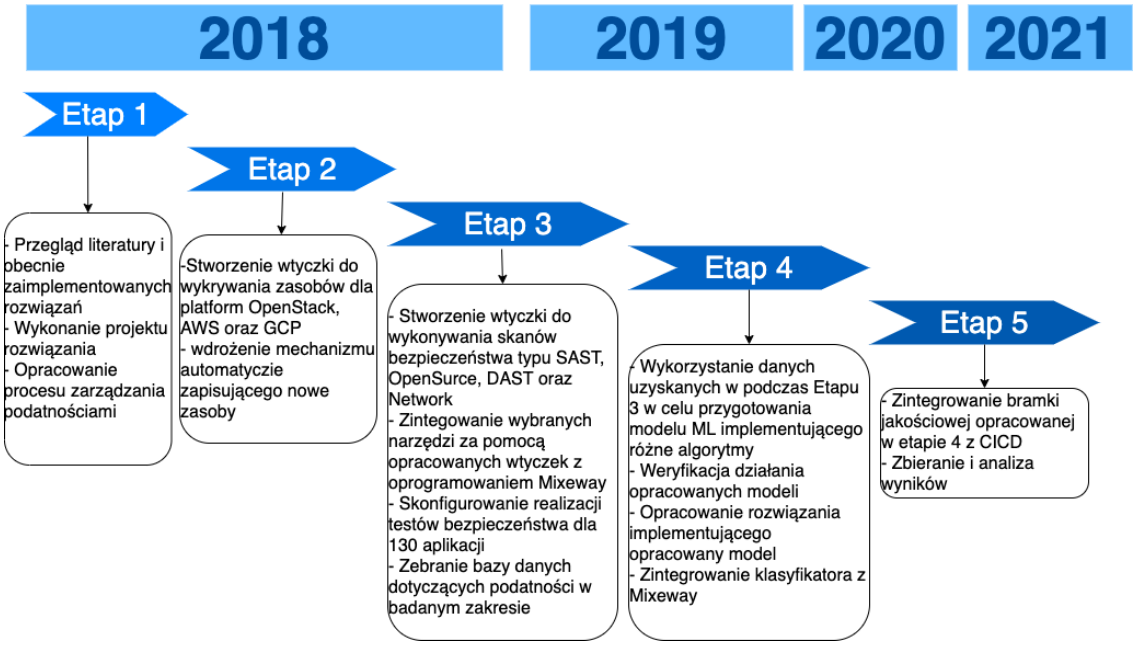
[31] `https://github.com/Mixeway/MixewayVulnAuditor`
[32] `https://hub.docker.com/repository/docker/mixeway/vulnauditor`



Rys. 5.8: Opis działania bramki bezpieczeństwa, która weryfikuje zgodność aplikacji z polityką bezpieczeństwa

5.2. Etapy wdrożenia rozwiązania

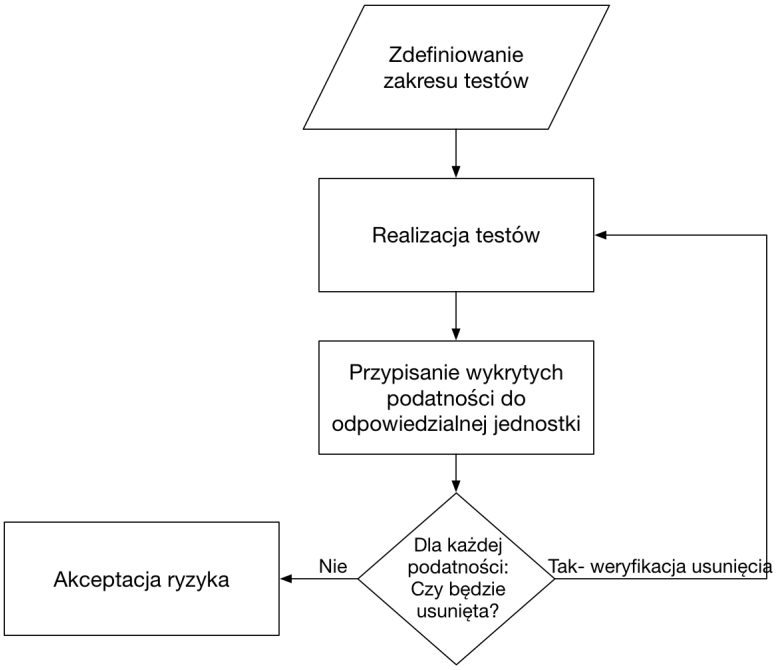
Rozwiązanie opisane w rozdziale 5.1 zostało wdrożone w infrastrukturze operatora telekomunikacyjnego Orange Polska. Z uwagi na dynamikę rozwoju technik związanych z wytwarzaniem oprogramowania oraz nienadążającymi za nimi sposobami zabezpieczeń aplikacji wdrożenie zostało podzielone na etapy (harmonogram przedstawiony na Rys. 5.9).



Rys. 5.9: Harmonogram wdrożenia Mixeway

5.2.1. Etap 1: Opracowanie procedury zarządzania podatnościami

Procesy związane z wykonywaniem testów bezpieczeństwa aplikacji są skomplikowaną operacją, która nie kończy się w momencie powstania raportu z testów zawierającym listę podatności bezpieczeństwa. Konieczne jest określenie zasad i obowiązków konkretnych osób (ról), które występują w procesie dostarczania oprogramowania. Istotne jest to, aby było wiadomo, kto powinien odebrać informację dotyczącą wykrycia nowego błędu tak, aby mógł on zostać szybko usunięty. Pierwszy etap wdrożenia skończył się opracowaniem i wdrożeniem procedury zarządzania podatnościami, która dokładnie określa podział ról i obowiązków między zespołami w celu szybkiego reagowania i usuwania wykrytych błędów bezpieczeństwa.



Rys. 5.10: Kroki realizowane zgodnie z przyjętą procedurą zarządzania podatnościami

Rys. 5.10 przedstawia operacje, które wykonywane są zgodnie z przyjętą procedurą. Po zdefiniowaniu zakresu testów oraz wygenerowaniu raportu zawierającego listę podatności następuje zgłoszenie wykrytych problemów jednostkom, które odpowiadają za daną aplikację lub system teleinformatyczny, którego dotyczy błąd. Następnie w ramach tych zespołów (które są adresatem raportu) następuje analiza możliwości usunięcia podatności lub weryfikacji tego, czy zgłoszony problem stwarza realne zagrożenie bezpieczeństwa. Gdy, w ramach wykonanej analizy została podjęta decyzja o usunięciu defektu, następuje powtórzenie testów w celu weryfikacji. W przeciwnym przypadku (np. w sytuacji, gdy koszt usunięcia podatności jest bardzo duży lub z analizy wynika, że zgłoszenie jest fałszywym alarmem) konieczna jest akceptacja ryzyka przez właściciela biznesowego systemu.

5.2.2. Etap 2: Wdrożenie modułu detekcji zasobów

Aby odpowiednio zarządzać bezpieczeństwem projektów realizowanych na platformach IaaS, gdzie zasoby zmieniają się dynamicznie (w ciągu godziny może powstać kilka nowych maszyn wirtualnych, podczas gdy kilka następnych zostanie usuniętych), konieczne jest posiadanie informacji dotyczących tego, jakie zasoby są w danym momencie w użyciu. Informacje te wykorzystywane są po to, aby możliwe było skonfigurowanie testu podatności. Bez dokładnej wiedzy dotyczącej tego ile i jakich zasobów w danym momencie działa operacyjnie konieczne byłoby definiowane zakresu testu w formie szerokiej klasy adresacji np. 10.0.0.0/16 zawierającym 65,534 adresów IP (podczas gdy rzeczywiście maszyn może być kilka lub kilkanaście). Nietrudno sobie wyobrazić różnicę w czasie trwania testu dla wymienionych wcześniej zakresów.

Dodatkowym elementem, który jest istotny do monitorowania, jeśli chodzi o bezpieczeństwo platform IaaS, są reguły sieciowe. Zgodnie z zasadą najmniejszych możliwych uprawnień zasoby takie jak aplikacje czy maszyny wirtualne powinny mieć możliwość komunikacji sieciowej wyłącznie z niezbędnymi zasobami. Reguła sieciowa określana jest przez: źródło (ang. Source) i miejsce docelowe (ang. Destination), port oraz protokół transportowy (TCP/UDP). Aby możliwa była odpowiednia weryfikacja tego jakie połączenia są dopuszczalne wewnątrz platformy IaaS opracowana została polityka bezpieczeństwa dla komunikacji sieciowej:

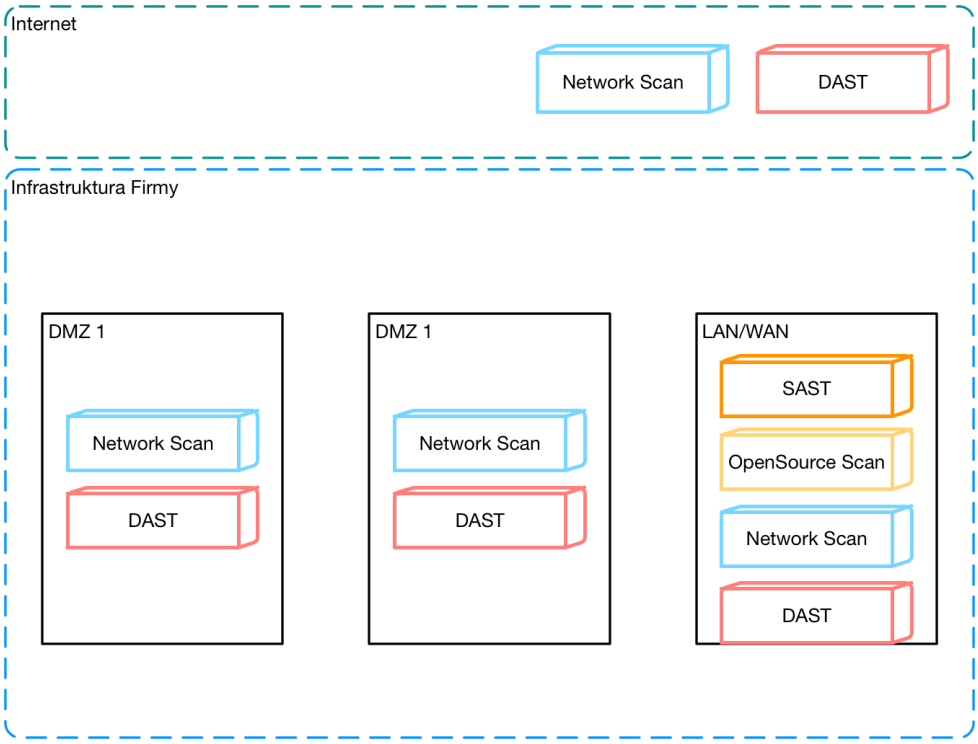
- Niedozwolone jest definiowanie reguły umożliwiającej dostęp z każdego zasobu do każdego celu (tzw. ANY:ANY),
- Niedozwolona jest komunikacja między zasobami korzystając z portu 22 (protokół SSH). Wyjątkiem jest komunikacja inicjowana przez element pełniący rolę bastion host'a,
- Niedozwolona jest komunikacja protokołami bazodanowymi między zasobami należącymi do różnych systemów/projektów.

Wdrożenie etapu 2 zakończyło się możliwością pobierania (w trybie ciągłym) informacji o działających zasobach oraz weryfikowaniu definicji reguł sieciowych na platformach OpenStack, AWS oraz GCP.

5.2.3. Etap 3: Wdrożenie modułu zarządzania testami bezpieczeństwa

Wdrożenie etapu 2 umożliwiło zebranie danych dotyczących zakresu, jaki musi zostać poddany testom podatności. Jednakże, przed przejściem do stanu, w którym możliwe jest automatyczne uruchamianie skanów, konieczny był krok związany z instalacją i konfiguracją infrastruktury, na której realizowane będą testy.

Rys. 5.11 przedstawia infrastrukturę realizującą testy w sieci firmy. Segmentacja sieci uniemożliwia bezpośrednie połączenia między wydzielonymi strefami (DMZ1, DMZ2 oraz LAN/WAN). Z tego powodu konieczne jest umieszczenie dedykowanego oprogramowania pozwalającego na wykonywanie testów sieciowych oraz DAST w każdym segmencie



Rys. 5.11: Umiejscowienie różnych skanerów podatności w infrastrukturze organizacji

sieci. Skanery SAST oraz OSA (ang. Open Source Analysis) bazują na kodzie źródłowym pobranym z repozytorium, przez co nie ma potrzeby na ich multiplikowanie.

Następnym krokiem było opracowanie i wdrożenie wtyczek pozwalających na zdalne sterowanie skanerami bezpieczeństwa: sieciowymi (Tenable Nessus i OpenVAS [104]), DAST (Acunetix i Burp [105]) oraz SAST (Fortify i Checkmarx [106]). Dzięki integracji umożliwiającej zdalne wykonywania operacji takich jak: definiowanie zakresu testu, uruchomienie skanu oraz pobranie wykrytych podatności bezpieczeństwa. Testy bezpieczeństwa mogły być wykonywane automatycznie bez potrzeby udziału człowieka. Doprowadziło to do powstania bazy danych zawierającej informację na temat ponad 50 000 podatności wykrytych w ten sposób. Szczegółowy opis działania tego modułu został przedstawiony w rozdziale 5.1.2.

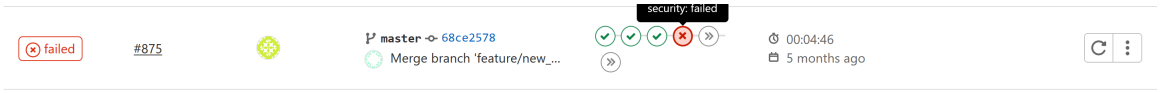
5.2.4. Etap 4: Opracowanie i wdrożenie modułu korelacji wyników

W efekcie realizacji poprzednich etapów stworzona została baza zawierająca informacje na temat wykrytych błędów bezpieczeństwa. Zebrano informacje o ponad 50 000 podatnościach. Aby możliwe było wykorzystanie zebranych danych w procesie nauki opracowywanego klasyfikatora, każdy rekord z zestawu danych został przeanalizowany. Bazując na polityce bezpieczeństwa organizacji, wiedzy i doświadczeniu, kontekście aplikacji (w której wykryto problem) oraz eksperymentalnych próbach wykorzystania konkretnych defektów, każdej próbce nadano wartości “1” jeśli jest ona konieczna do usunięcia “0” jeśli jest ona nieistotna.

Tak przygotowany zestaw danych został wykorzystany w kolejnym kroku, aby eksperymentalnie porównać skuteczność działania klasyfikatorów opracowanych z wykorzystaniem algorytmów Sieci Neuronowych, Lasu Losowego oraz SVM. W wyniku przeprowadzonej analizy najbardziej odpowiedni model został zaimplementowany oraz zintegrowany z systemem Mixeway, dzięki czemu możliwe jest określenie z odpowiednią dokładnością wagi dla nowo wykrywanych podatności bezpieczeństwa. Szczegółowy opis działania tego modułu został przedstawiony w rozdziale 5.1.3.

5.2.5. Etap 5: Wdrożenie bramki bezpieczeństwa w łańcuchu dostarczania oprogramowania

W ostatnim etapie wdrożona została bramka bezpieczeństwa, która wykorzystując dane uzyskane przez moduł korelacji wyników, błyskawicznie dostarcza informacje dotyczące stanu bezpieczeństwa konkretnej aplikacji. Informacja ta wykorzystywana jest w łańcuchu dostarczania oprogramowania, aby ustalić czy proces budowania aplikacji może być kontynuowany. Pozwala to odpowiednio wcześnie przerwać łańcuch w przypadku wykrycia istotnych błędów bezpieczeństwa oraz zgłoszenie ich do zespołu programistycznego.

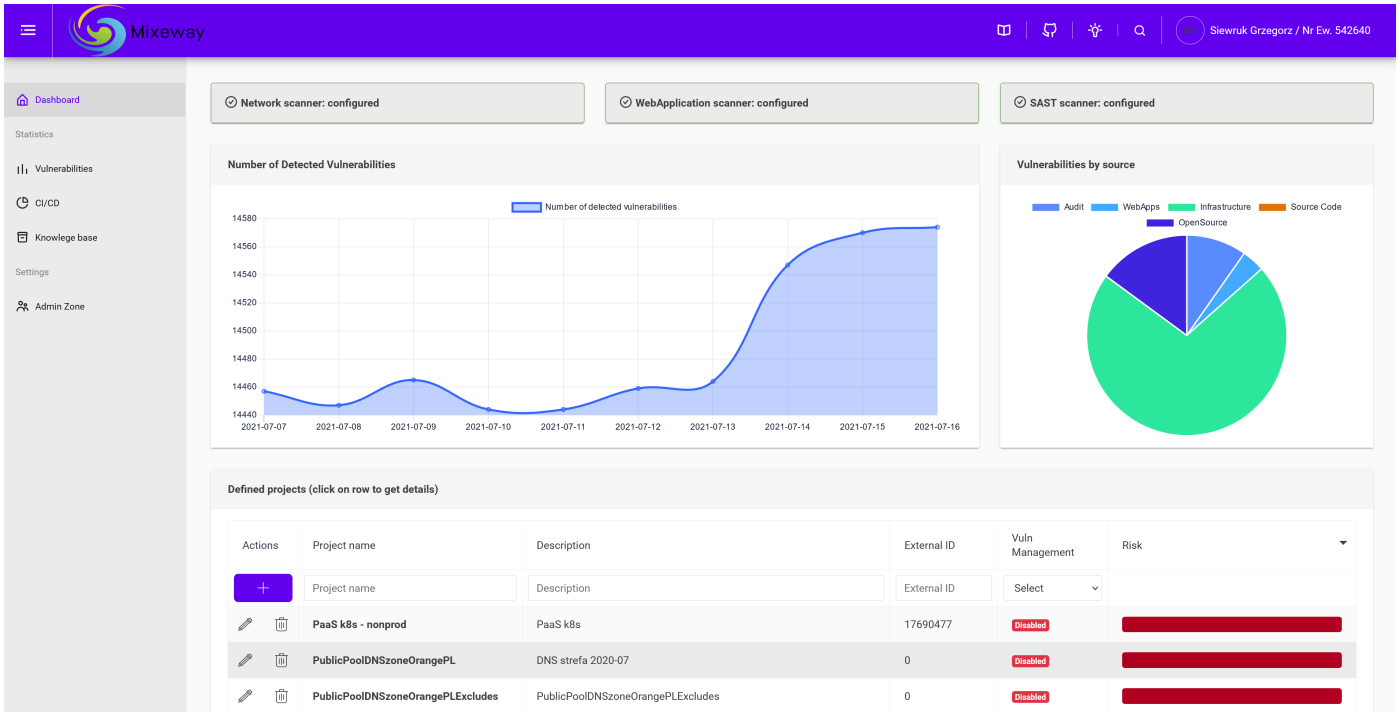


Rys. 5.12: Wynik działania mechanizmu Gitlab-CI w przypadku braku spełnienia polityki bezpieczeństwa

Rys. 5.12 przedstawia wynik działania mechanizmu CICD, który zawiera krok polegający na weryfikacji wyniku zwróconego przez bramkę bezpieczeństwa. W przedstawionym przykładzie wynik działania bramki wskazuje na niespełnienie polityki bezpieczeństwa w wyniku czego łańcuch dostarczania oprogramowania został przerwany, a kolejne zdefiniowane zadania (takie jak zbudowanie aplikacji oraz opublikowanie jej w odpowiednim rejestrze) nie zostały wykonane.

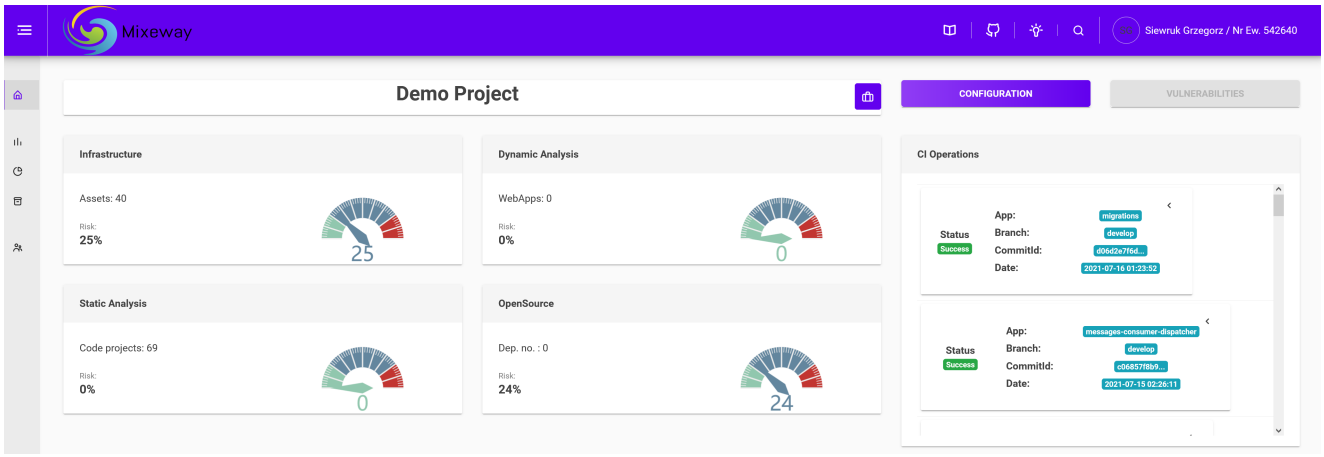
5.3. Przedstawienie graficznego interfejsu użytkownika

Opracowane rozwiązanie implementuje rozbudowany graficzny interfejs użytkownika, za pomocą którego użytkownicy mogą w prosty sposób przeglądać statystyki związane z bezpieczeństwem konkretnych projektów.

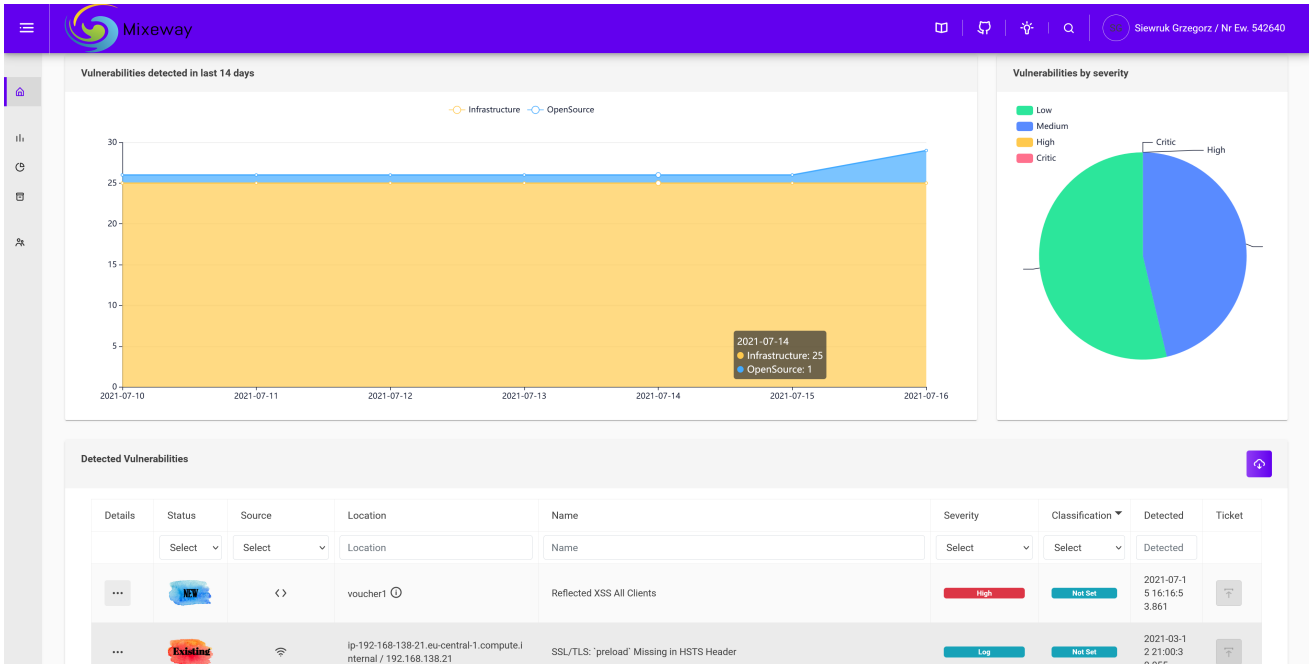


Rys. 5.13: Zrzut ekranu z głównego menu Mixeway

Rys. 5.13 przedstawia widok zaraz po zalogowaniu użytkownika. Menu z lewej strony zawiera możliwość przejścia do takich stref jak globalne statystyki podatności, globalne statystyki CICD, baza wiedzy czy strefa administratora. W głównej części widoczny jest status podłączonych skanerów podatności, poniżej którego widać statystyki dotyczące trendu w wykrytych podatnościach dla całej platformy. Na samym dole znajduje się tabela ze zdefiniowanymi projektami, które mogą posiadać wiele zasobów (takich jak aplikacje internetowe czy repozytoria kodu).



Rys. 5.14: Zrzut ekranu prezentującego konkretny projekt (część pierwsza)



Rys. 5.15: Zrzut ekranu prezentującego konkretny projekt (część druga)

Rys. 5.14 oraz 5.15 prezentują informację, które widoczne są po otwarciu konkretnego projektu. Na samej górze (Rys. 5.14) wyświetlane są zasoby, które zostały dodane do projektu. W przypadku przedstawianego zrzutu są to: 43 maszyny wirtualne (adres IP) oraz 69 repozytoria kodu. Po prawej stronie widać informacje o wyniku działania bramki bezpieczeństwa uruchomionej w procesie CICD ze szczegółami o tym jakiej aplikacji oraz gałęzi kodu dotyczy wraz z wynikiem (w obu widocznych przypadkach polityka bezpieczeństwa była spełniona - Success).

Poniżej (Rys. 5.15) prezentowane są wykresy, z przedstawionym trendem liczby podatności w projekcie w ciągu ostatnich 14 dni. Pod wykresami znajduję się tabela ze szczegółami dotyczącymi wykrytych podatności.

5.4. Stan przed wdrożeniem rozwiązania w środowisku operatora telekomunikacyjnego

Aby lepiej zobrazować wpływ jaki miało wdrożenie systemu Mixeway w infrastrukturze operatora telekomunikacyjnego wykonana została analiza dotycząca liczby wykonywanych testów z wykorzystaniem narzędzi automatycznych (Tabela 5.1) oraz zasobów, które takimi testami zostały objęte (Tabela 5.2).

Tabela 5.1: Średnia liczba testów podatności z wykorzystaniem automatycznych narzędzi wykonywana w skali roku (nie zawiera ręcznych testów penetracyjnych). Stan przed wdrożeniem systemu Mixeway.

Typ testu	Liczba wykonanych skanów	Średni czas dostarczenia wyników
Testy sieciowe	786 000	14 dni
Testy SAST	760	7 dni
Testy DAST	400	2 dni
Testy OpenSource	0	n.d.

Duża liczba testów sieciowych wynika z charakteru realizacji tego typu skanowania. Raz w miesiącu wykonywane były testy, których zakres obejmował sieć o masce /16. Zakres skanu składający się z 65 534 adresów IP powtarzany raz w miesiącu pozwolił na zrealizowanie 786 000 skanów podatności sieciowych. Długi czas, który można zaobserwować przy tej pozycji, wynika z dużej liczby adresów IP poddanych skanowaniu. Uruchomienie testu, którego zakresem jest tak duża liczba skanowanych zasobów powodowało udostępnienie raportu dopiero po 14 dniach. Ustalenie właściciela zasobu, który jest odpowiedzialny za usunięcie wykrytych podatności miał również wpływ na czas, który minął od uruchomienia do przekazania właścicielowi biznesowemu informacji o błędzie. W przypadku statycznych testów kodu źródłowego (SAST) wykonano 760 testów, których czas trwania trwał od kilku do kilkunastu godzin. Weryfikacja specjalisty wydłużyła średni czas dostarczenia raportu do 7 dni. Testy dynamiczne były wykonywane 400 razy przy czasie dostarczenia raportu wynoszącym 2 dni, natomiast skanowanie bibliotek wykorzystywanych w projektach nie było wykonywane w ogóle.

Tabela 5.2: Zasoby objęte testami podatności. Stan przed wdrożeniem Mixeway.

Typ zasobu	Liczba
Maszyna wirtualna/serwer fizyczny	65 534
Aplikacja Internetowa (adres URL)	350
Repozytorium kodu źródłowego	40

Ręczna konfiguracja rozwiązań realizujących skanowanie, zapewnienie łączności sieciowej (konfiguracja urządzeń filtrujących sieć) oraz potrzeba uzyskania uprawnień do repozytorium kodu źródłowego znacząco wpłynęła na liczbę zasobów, które były objęte

testami podatności (Tabela 5.2). Potrzeba wykonania wymienionych operacji ograniczyła zasoby objęte skanami bezpieczeństwa do: 65 534 adresów IP (liczba adresów znajdującej się w sieci o masce /16), 350 aplikacji internetowych (adresów URL) oraz 40 repozytoriów kodu źródłowego.

5.5. Stan po wdrożeniu rozwiązania w środowisku operatora telekomunikacyjnego

Wdrożenie Mixeway umożliwiło realizację zadań związanych z wykonywaniem testów podatności w inny, bardziej zautomatyzowany sposób. Tabela 5.3 przedstawia średnią liczbę testów (w skali roku) wykonywanych przy wykorzystaniu opisywanego rozwiązania, podczas gdy Tabela 5.4 opisuje jak zmienił się zakres wykonywanego skanowania.

Tabela 5.3: Średnia liczba testów podatności z wykorzystaniem automatycznych narzędzi wykonywana w skali roku (nie zawiera ręcznych testów penetracyjnych). Stan po wdrożeniu Mixeway.

Typ testu	Liczba wykonanych skanów	Średni czas dostarczenia wyników
Testy sieciowe	800 000	3h
Testy SAST	25 200	30min
Testy DAST	14 000	5h
Testy OpenSource	25 200	2min

Wykorzystując nowe podejście (oparte o Mixeway) możliwe jest dokładne określenie, które zasoby sieciowe będą poddane testom powtarzanym co 3 dni (zmiana z testów wykonywanych raz w miesiącu), jest to możliwe, ponieważ mniejszy zakres skanowania powoduje skrócenie czasu jego realizacji. Pozwoliło to także na zmniejszenie czasu dostarczenia wyników do 3 godzin co stanowi skrócenie czasu o ponad 99% w stosunku do stanu sprzed wdrożenia. W przypadku testów skanerami DAST, SAST oraz OpenSource widać również znaczny wzrost liczby wykonywanych testów przy jednoczesnym skróceniu czasu przekazania wyników. Statyczne testy aplikacji są wykonywane 33 razy częściej a czas dostarczenia wyników jest o ponad 99% krótszy. Testy dynamiczne realizowane są 35 razy częściej podczas gdy raporty udostępnione zostają 90% szybciej. Spowodowane jest to tym, że konfiguracja nie jest wykonywana przez człowieka, a jest zlecana przez narzędzia CICD. Weryfikacja jest wykonywana przez opracowany algorytm zaimplementowany w module korelacji wyników (rozdział 5.1.3).

W rezultacie zastosowania Mixeway liczba zasobów sieciowych objętych testami znacznie zmalała, należy jednak pamiętać, że w przypadku testowania pełnych klas adresowych nie wszystkie adresy IP są przypisane do serwera czy maszyny wirtualnej. W wyniku tej zmiany testy realizowane są znacznie szybciej przy jednoczesnym objęciu skanami wszystkich zasobów, które są udostępniane w organizacji. Zdalne sterowanie konfiguracją testów umożliwiło zatem zwiększenie liczby aplikacji internetowych oraz repozytoriów

Tabela 5.4: Zasoby objęte testami podatności. Stan po wdrożeniu Mixeway.

Typ zasobu	Liczba
Maszyna wirtualna/serwer fizyczny	8 000
Aplikacja Internetowa (adres URL)	3 000
Repozytorium kodu źródłowego	140

kodu źródłowego, które są poddawane regularnym i automatycznym testom podatności. Dzięki takiemu podejściu możliwe jest wykrywanie potencjalnych problemów bezpieczeństwa krótko po ich powstaniu, co znacznie zmniejsza ryzyko wykorzystania defektu przez atakującego. W szczególnym przypadku wystąpienie podatności CVE-2021-34527 pozwalającej na zdalne przejęcie kontroli nad kontrolerem domeny po wdrożeniu Mixeway (w najgorszym przypadku) zostanie wykryta po 3 dniach (interwał czasowy powtarzania testów sieciowych), podczas gdy przed wdrożeniem byłoby to 30 dni.

Dzięki wdrożeniu Mixeway w infrastrukturze operatora telekomunikacyjnego możliwe było wykrycie, a następnie usunięcie wielu podatności bezpieczeństwa. Liczba realizowanych testów SAST oraz DAST wzrosła ponad 30 razy (w stosunku do stanu przed wdrożeniem - por. Tabele 5.1 oraz 5.3), a czas dostarczania wyników dla tego typu skanów skrócił się o ponad 90%. W wyniku wdrożenia stworzono także unikalną bazę danych zawierającą informacje o wykrytych błędach wraz z kontekstem wykorzystywania systemu oraz analizą czy defekt jest istotny, czy nie. Opisane wcześniej akcje umożliwiły wykonanie eksperymentu polegającego na weryfikacji skuteczności różnych algorytmów uczenia maszynowego w celu klasyfikacji podatności bezpieczeństwa. Szczegółowy przebieg eksperymentu oraz jego wyniki opisane zostały w dalszej części niniejszej rozprawy.

6. Metodyka badawcza

Łącząc wyzwania, które zostały opisane w rozdziale 3.4, czyli zapewnienie bezpieczeństwa w procesie CICD i uwzględnienie wyników testów podatności w procesach decyzyjnych dotyczących publikacji nowych wersji aplikacji oraz wytypowane algorytmy uczenia maszynowego przedstawione w rozdziale 4 przeprowadzony zostały badania eksperymentalne, których celem jest:

- *Opracowanie klasyfikatorów* stworzonych w oparciu na modelach przygotowanych z wykorzystaniem algorytmów SVM, Sieci Neuronowe oraz Las Losowy.
- *Porównanie wartości metryk*: precyzji, dokładności, wartości współczynnika F1, oraz czułości dla opracowanych klasyfikatorów.
- *Stworzenie prototypu aplikacji* implementującej wybrany algorytm, która będzie działała jako jeden z komponentów systemu Mixeway (moduł korelacji wyników opisany w rozdziale. 5.1.3)

Badania eksperymentalne zostały przeprowadzone na komputerze z następującą konfiguracją: CPU Intel Core i7-7700HQ, 32GB RAM, GPU NVIDIA GeForce GTX 1050. Aplikacja realizująca zestaw badań została przygotowana z wykorzystaniem języka programowania Python (wersja 3) oraz bibliotekami TensorFlow[33] oraz scikit-learn[34].

6.1. Sposób przeprowadzenia badań eksperymentalnych

Konkretne kroki, które zostały wykonane podczas przeprowadzania badań, przedstawiono na Rys. 6.1. Pierwszy krok polegający na zebraniu odpowiednio dużego i zróżnicowanego zestawu danych trwał 12 miesięcy i był realizowany podczas drugiego etapu wdrożenia Mixeway (opisanego w rozdziale 5.2). Etap drugi polegał na przygotowaniu eksportu danych w odpowiednim formacie zawierającym wymagane informacje. Warto zauważyć, że omawiany zestaw danych został pobrany z kilku niezależnych narzędzi (skanerów podatności), z których każdy posiada unikalny format opisywania wykrytych podatności bezpieczeństwa. Wynikiem tej operacji jest zestaw danych przedstawiony w podrozdziale 6.2. Kolejnym etapem jest przetwarzanie wstępne zestawu danych, które opisano dokładnie w podrozdziale 6.3. Krok czwarty zakłada wykorzystanie przygotowanego zestawu danych w celu wytrenowania modelu. Każdy z badanych algorytmów zawiera inne parametry konfiguracyjne, które wymagają dostosowania (i zweryfikowania ich wpływu na wynik działania klasyfikatora). Dobrane parametry zostały przedstawione i opisane w podrozdziale 6.4. Ostatni etap to weryfikacja wyników przedstawiona w podrozdziale 7.

[33] https://tensorflow.org
[34] https://scikit-learn.org

Rys. 6.1: Etapy metodyki badań eksperymentalnych

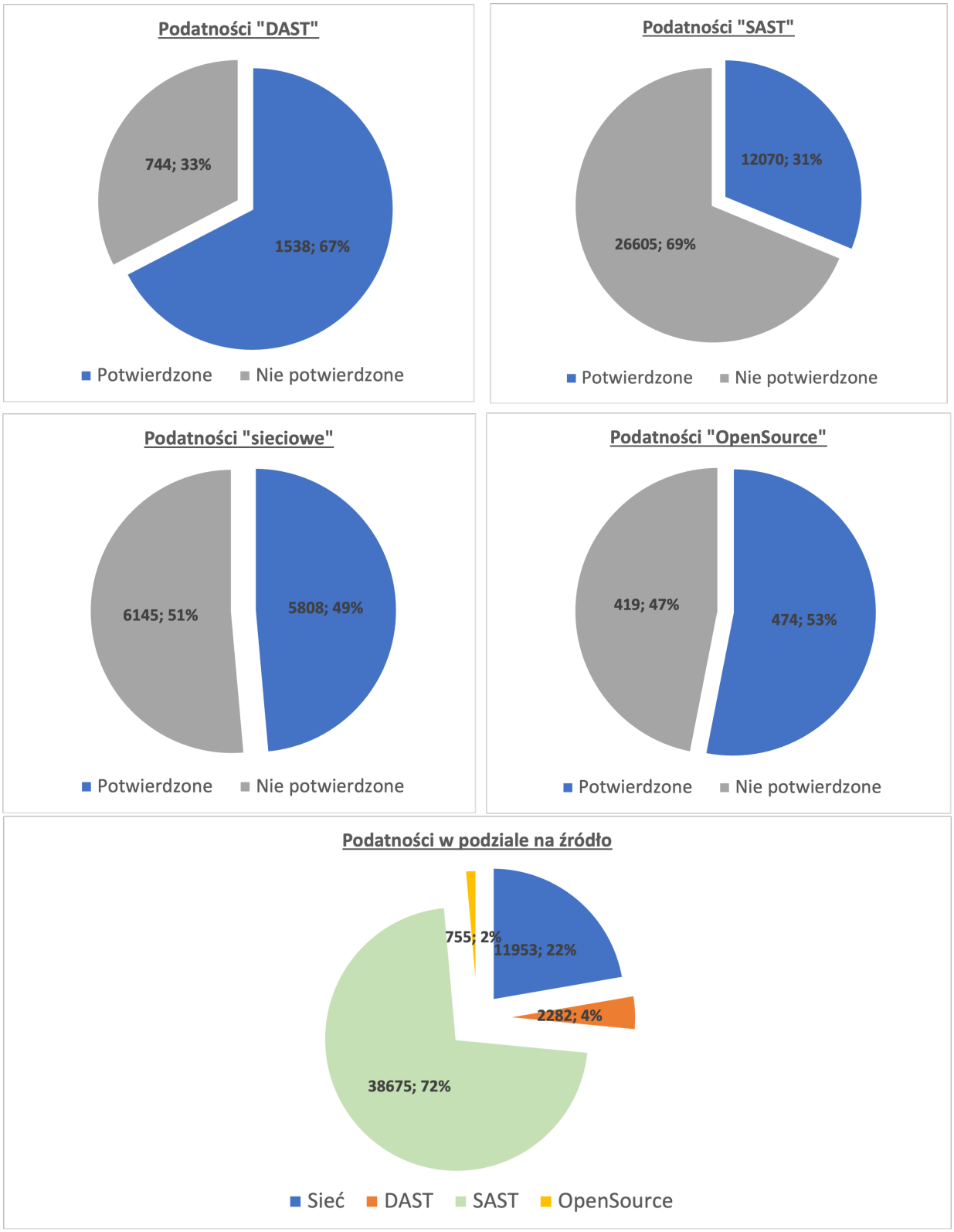
6.2. Opis danych

Zestaw danych treningowych składa się z 53 665 podatności bezpieczeństwa wykrytych w systemach teleinformatycznych znajdujących się w infrastrukturze operatora telekomunikacyjnego. Rozkład zasobów, w których znaleziono i zgłoszono podatności bezpieczeństwa przedstawiono w Tabeli 5.4. Podatności sieciowe dotyczą maszyn wirtualnych lub serwerów fizycznych. Podatności SAST oraz OpenSource dotyczą repozytoriów kodu źródłowego, natomiast podatności DAST dotyczą aplikacji internetowych.

Wszystkim wykrytym podatnościom nadano etykietę ([107]), która przyjmuje jedną z dwóch klas - "CRV" czyli podatność istotna i wymagająca poprawy oraz "DRNV" czyli podatność zgłoszona przez narzędzia, ale nieistotna z punktu widzenia kontekstu, w którym została wykryta.

Rys. 6.2 przedstawia szczegółowy rozkład wartości "CRV" oraz "DRNV" w zebranym zestawie danych. Z przedstawionych wykresów wynika, że skanery typu DAST charakteryzują się dużą dokładnością zgłaszanych naruszeń. Niemal 65% wykrytych podatności zostało ocenionych jako wymagające poprawy. Na drugim biegunie znajdują się skanery typu SAST, gdzie wyniki są niemal odwrotne. W tym przypadku tego typu skanerów 69% zgłoszonych błędów jest nieistotna. Znaczące różnice wynikają ze sposobu działania tego typu urządzeń [108]. Bardzo często scenariusz testów dynamicznych (logika aplikacji) zakłada krok, który ma na celu potwierdzenie wystąpienia podatności bezpieczeństwa oraz przedstawienie dokładnego sposobu na jego odtworzenie. Natomiast w przypadku testów statycznych nie ma takiej możliwości, gdyż skaner dysponuje wyłącznie kodem źródłowym. Testy sieciowe oraz weryfikacje bezpieczeństwa wykorzystywanych bibliotek prezentują podobny rozkład błędów wymagających poprawy oscylujący w granicach 50%. Ostatni wykres przedstawia rozkład źródeł, z których pochodzą podatności. Narzędzia SAST dostarczyły dane na temat 72% błędów, skanery sieciowe na temat 27%, dynamiczna analiza 4% natomiast weryfikacja zależnych bibliotek 2%. Widoczne dysproporcje w zebranym zbiorze danych (dotyczące źródła pochodzenia defektów) nie mają wpływu na wynik eksperymentu, co wynika z normalizacji formatu każdego rekordu znajdującego się w zestawie danych (niezależnie od źródła, wszystkie podatności zawierają ustalone i te same pola takie jak: nazwa podatności, kontekst czy opis).

Tabela 6.1 opisuje atrybuty danych, z których składa się każdy rekord w zbiorze danych treningowych. Z uwagi na to, że informacje dostarczane przez różne narzędzia nie są



Rys. 6.2: Podział zestawu danych na podatności potwierdzone i niepotwierdzone (cztery wykresy z góry) oraz liczba wykrytych podatności w podziale na źródła (wykres na dole).

Tabela 6.1: Atrybuty zawarte w zestawie danych

Nazwa atrybutu	Opis
Nazwa aplikacji	Nazwa zasobu, którego testy doprowadziły do wykrycia błędu. W przypadku wykrycia podatności sieciowych przyjmuje wartość nazwy urządzenia, w przypadku testów DAST przyjmuje wartość URL testowanej aplikacji internetowej, natomiast w przypadku testów SAST oraz OpenSource przyjmuje wartość nazwy repozytorium kodu źródłowego.
Kontekst	Kontekst opisuje sposób wykorzystania zasobu. Przykładowy kontekst zawiera informacje dotyczące przeznaczenia aplikacji, warstwy wykrycia lub krytyczności biznesowej.
Nazwa podatności	Nazwa podatności pobrana bezpośrednio ze skanera wykonującego testy.
Opis podatności	Opis podatności pobrany bezpośrednio ze skanera wykonującego testy. Pole to jest wzbogacane o dodatkowe informacje, w zależności od aplikacji, która jest źródłem danych. Dodatkowe dane to na przykład: dowód wystąpienia błędu, rekomendacje naprawy oraz dane dotyczące odtworzenia podatności.
Krytyczność	Krytyczność pobrana bezpośrednio ze skanera wykonującego testy.
Etykieta	Nadana przez autora niniejszej rozprawy wartość (powstała w wyniku ręcznej analizy każdego rekordu), determinująca to czy podatność wymaga poprawy, czy jest nieistotna.

jednakowe, zbiór danych musiał zostać zunifikowany. Idealnym przykładem tej operacji jest pole “Opis podatności”. Wszystkie skanery podatności udostępniają informacje dotyczącą dodatkowych informacji na temat wykrytej podatności tj. wysokopoziomowy opis wektora ataku oraz jego wpływ na ciągłość działania. Niektóre z rozwiązań dodatkowo udostępniają także dane będące rekomendacjami jeśli chodzi o sposób usunięcia podatności oraz informacji na temat tego, w jaki sposób odtworzyć zgłoszony błąd (żeby potwierdzić jego występowanie). Wspomniane pole zawiera wszystkie dostępne dane dodatkowe, w zależności od tego czy źródło je udostępnia.

Tabela 6.2: Wartości pola kontekst

Nazwa	Możliwe wartości
Sposób wykorzystania	internal, external
Miejsce uruchomienia	local data center, remote data center
Podatna warstwa	infrastructure, web application, source code
Docelowy użytkownik	customer, emoloyee
Krytyczność biznesowa zasobu	sox, pci dss, critical or not critical

Unikalnym polem, które odróżnia zestaw danych wykorzystany w opisywanych badaniach eksperymentalnych od tych umieszczonych w rozdziale 2 jest kontekst, w którym wykorzystano badany zasób. Możliwe wartości tego pola zostały przedstawione w Tabeli 6.2. Każda podatność w zestawie danych treningowych zawiera przynajmniej jedną informację dotyczącą kontekstu, w jakim błąd został wykryty. Wartości te zostały nadane

na etapie definiowania zakresu testów bezpieczeństwa. Sposób wykorzystania opisuje to czy badana aplikacja jest przeznaczona do użytku wewnętrznego w organizacji, czy też będzie wykorzystywana przez klientów zewnętrznych. Z kolei miejsce uruchomienia opisuje, na jakiej infrastrukturze działa dany system - zarządzanej przez organizację lub znajdującej się w chmurze publicznej. Podatna warstwa definiuje natomiast, który ze skanerów podatności jest źródłem informacji. Ostatnie pole kontekstu to krytyczność biznesowa aplikacji, opisująca wpływ systemu na ciągłość działania usług lub potrzebę jego dostosowania do standardów takich jak SOX [109] lub PCI DSS [110].

Biorąc pod uwagę charakter informacji, które zawarte są w przygotowanej bazie danych, a które w dużym zakresie przyjmują format opisów konieczne jest wykonanie szeregu operacji w ramach przetwarzania wstępnego tak, aby możliwe było przekazanie rekordu (o formacie opisanym powyżej)

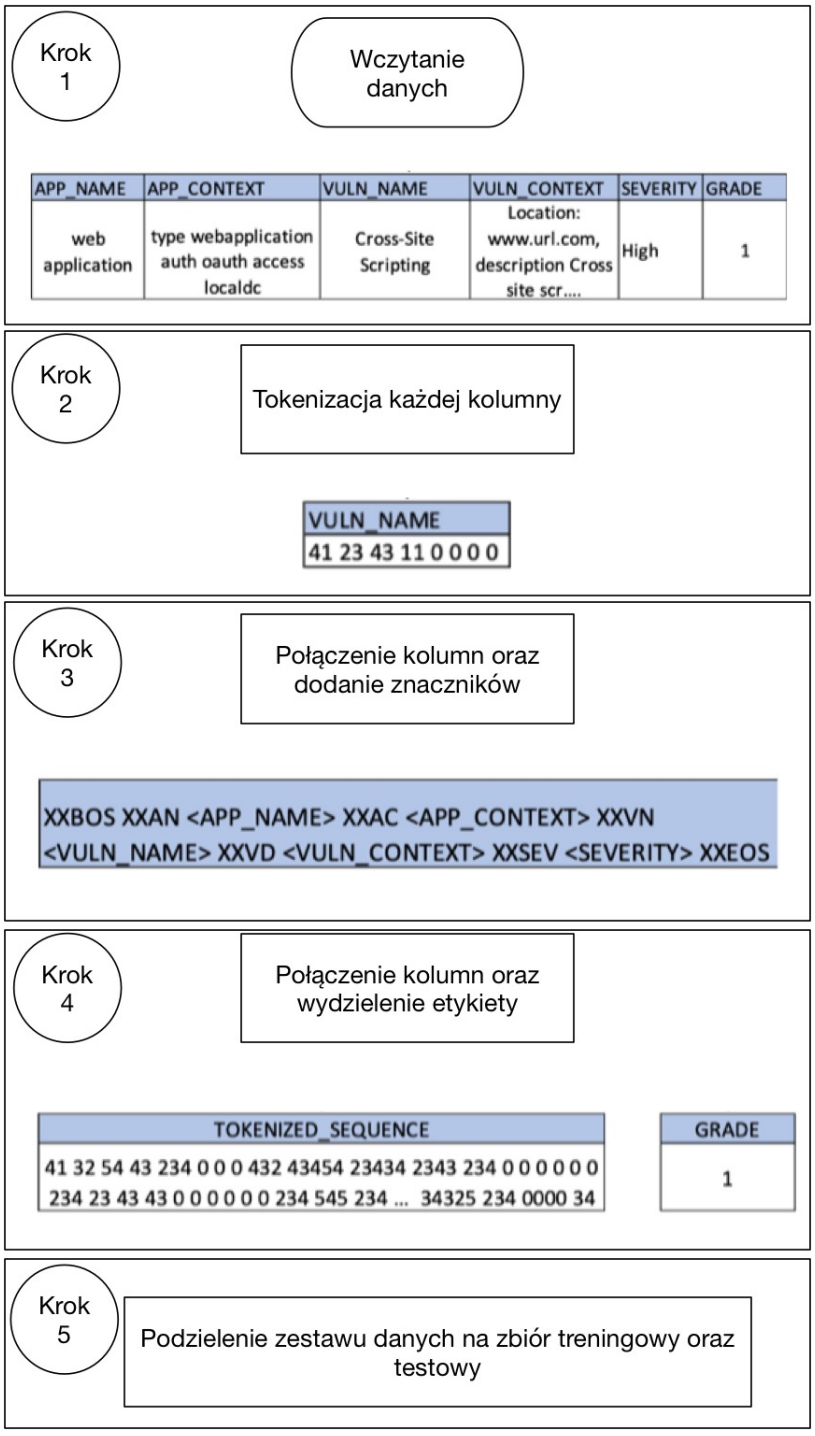
Informacje opisujące podatności bezpieczeństwa w dużym zakresie przyjmują format opisowy. Aby taki zestaw danych mógł zostać przekazany do procesu nauki modelu klasyfikatora, konieczne jest wykonanie szeregu operacji w ramach przetwarzania wstępnego, w wyniku którego powstaje zestaw danych treningowych oraz testowych pozwalających na zweryfikowanie skuteczności opracowanego algorytmu.

6.3. Przetwarzanie wstępne

Aby przygotowany model klasyfikatora cechował się wysoką dokładnością, niezbędne jest odpowiednie przeprowadzenie przetwarzania wstępnego zestawu danych treningowych [111]. W celu zachowania charakteru danych, którymi są informacje o podatnościach bezpieczeństwa, przetwarzanie wstępne składa się z pięciu etapów przedstawionych na Rys. 6.3.

W pierwszym kroku wykonywane są operacje związane z wczytaniem całego zestawu danych do pamięci komputera. W wersji aplikacji, która jest udostępniona za pośrednictwem serwisu GitHub, wymagany jest zestaw danych w formacie *CSV*.

Krok drugi to tokenizacja danych (jedna z technik przetwarzania języka naturalnego). W wyniku tego działania każda kolumna, która składała się z ciągów znaków, zamieniana jest na wektor o wymiarze $[1, N]$, gdzie N to liczba słów, z których składała się dana kolumna. Należy wziąć pod uwagę także to, że liczba słów w kolumnach takich jak: “nazwa podatności”, “kontekst aplikacji”, czy “opis podatności” może się znacznie różnić natomiast rozmiar macierzy, przechowującej tokenizowane dane musi być znany i jednakowy dla wszystkich przykładów. Aby rozwiązać ten problem, określony został rozmiar macierzy dla każdej kolumny – $M_{kolumny}$. Jeśli dane pole zawiera mniej słów, niż $M_{kolumny}$, uzupełniane jest o tyle cyfr 0 ile wynosi różnica $M - N$. Drugim przypadkiem jest sytuacja gdy $N > M$ – wtedy nadmiarowe słowa są pomijane. W wyniku opisanego powyżej procesu utracie ulega część danych. Aby zminimalizować wpływ tej operacji na dokładność działania klasyfikatora wartości $M_{kolumny}$ zostały dobrane tak, aby 90%



Rys. 6.3: Etapy, z których składa się przetwarzanie wstępne w prowadzonych badaniach eksperymentalnych

danych z zestawu treningowego nie wymagało utraty części informacji. Ostatecznym wynikiem etapu przetwarzania wstępnego jest zestaw danych przedstawiony w formie dwuwymiarowej macierzy zawierającej liczby całkowite. Format, w jakim przedstawiony jest konkretny przykład ze zbioru danych może zostać opisany jako zbiór macierzy o rozmiarach [1..$M_{NazwaAplikacji}$] [1..$M_{KontekstAplikacji}$] [1..$M_{\text{NazwaPodatności}}$] [1..$M_{\text{OpisPodatności}}$] [1] [1], w których dwie ostatnie kolumny odpowiadają kolejno krytyczności i etykiecie, które zawsze zawierają jednowyrazową wartość.

Następnym krokiem jest połączenie kolumn w jedną macierz o wymiarach JxK, gdzie J to liczba przykładów w zbiorze danych natomiast K to liczba elementów, które znajdują się w każdym przykładzie. Aby umożliwić budowanemu klasyfikatorowi łatwiejsze zrozumienie, gdzie kończy się konkretne pole, a gdzie zaczyna kolejne, pomiędzy poszczególne kolumny wstawiono znaczniki. Wartości znaczników zostały dobrane tak, aby szansa na ich wystąpienie w zestawie danych była minimalna. Wartości, które określają początki kolumn zostały przedstawione w Tabeli 6.3. Wykonanie tej operacji, a następnie połączenie kolumn w jedną macierz powoduje uzyskanie zbioru danych, który może zostać przedstawiony w ogólnej formie: [$XXBOS$ $XXAN$ 1..$M_{NazwaAplikacji}$ $XXAC$ 1..$M_{KontekstAplikacji}$ $XXVN$ 1..$M_{\text{NazwaPodatności}}$ $XXVD$ 1..$M_{\text{OpisPodatności}}$ $XXSEV$ 1 $XXEOS$]

Tabela 6.3: Znaczniki definiujące początek konkretnych pól w zestawie danych treningowych

Nazwa znacznika	Opis
XXBOS	Znacznik określający początek konkretnego przykładu.
XXAN	Znacznik określający początek pola zawierającego nazwę aplikacji.
XXAC	Znacznik określający początek pola kontekstu, w którym została wykryta podatność.
XXVN	Znacznik określający początek pola zawierającego nazwę wykrytej podatności.
XXVD	Znacznik określający początek pola zawierającego opis podatności.
XXSEV	Znacznik określający początek pola zawierającego krytyczność podatności.
XXEOS	Znacznik definiujący koniec konkretnego przykładu.

Krok czwarty ma na celu oddzielenie etykiety od stokenizowanej sekwencji danych, w wyniku czego powstają dwie macierze. Ostatnia operacja polega na podzieleniu zbioru danych na dane treningowe (czyli takie, które zostaną wykorzystane do wytrenowania modelu) i dane testowe (czuli takie, które zostaną wykorzystane w celu weryfikacji skuteczności opracowanego modelu). Zbiór treningowy zostanie wykorzystany podczas opracowywania modelu z wykorzystaniem konkretnego algorytmu uczenia maszynowego, którego dokładność zostanie zweryfikowana poprzez sprawdzenie wyniku działania klasyfikacji na zestawie danych testowych [112]. W szczególności ta ostatnia operacja jest

niezwykle istotna. W sytuacji, w której zestaw danych treningowych zawierać będzie mało zróżnicowane próbki (np. zawierające tylko definicje podatności dla jednej aplikacji lub pochodzące tylko z jednego źródła) bardzo prosto doprowadzić do sytuacji przetrenowania modelu [113]. Sytuację tę można rozpoznać, jeśli opracowany model charakteryzuje się wysoką dokładnością predykcji dla danych treningowych, jednak osiąga zdecydowanie gorsze wyniki dla danych testowych. Wykorzystywana podczas eksperymentu biblioteka (TensorFlow) posiada zaimplementowanych kilka metod, które pozwalają na odpowiednio zróżnicowany podział zestawu danych [114], dzięki czemu wystąpienie opisanego powyżej przypadku może mieć jedynie bardzo ograniczony wpływ na uzyskane rezultaty.

6.4. Opis hiperparametrów badanych klasyfikatorów

Większość algorytmów uczenia maszynowego posiada dwa typy parametrów. Pierwszy to wartości, które są możliwe do obliczenia lub oszacowania na podstawie danych (na przykład waga dla konkretnego neuronu w Sieci Neuronowej). Takie zmienne nazywane są parametrami. Drugi typ to wartości, których nie można oszacować. W większości przypadków zmienne te są ustalane za pomocą heurystyki lub dostrajane dla konkretnego problemu (na przykład liczba drzew w Lesie Losowym lub liczba neuronów w warstwie dla Sieci Neuronowych) [115]. Zmienne tego typu nazywane są hiperparametrami algorytmu.

Wartości hiperparametrów wpływają w bardzo dużym stopniu na skuteczność działania budowanego klasyfikatora [116]. W przedstawianych badaniach eksperymentalnych wartości omawianych zmiennych zostały dobrane za pomocą dostrajania. Wpływ zmiany danego parametru na dokładność działania predykcji nie jest przedmiotem niniejszej rozprawy. Listę hiperparametrów, które wykorzystywane są w poszczególnych algorytmach przedstawiono dla SVM w Tabeli 6.4, dla Sieci Neuronowych w Tabeli 6.5 oraz dla Lasu Losowego w Tabeli 6.6.

Tabela 6.4: Wybrane hiperparametry wykorzystywane w algorytmie SVM [117]

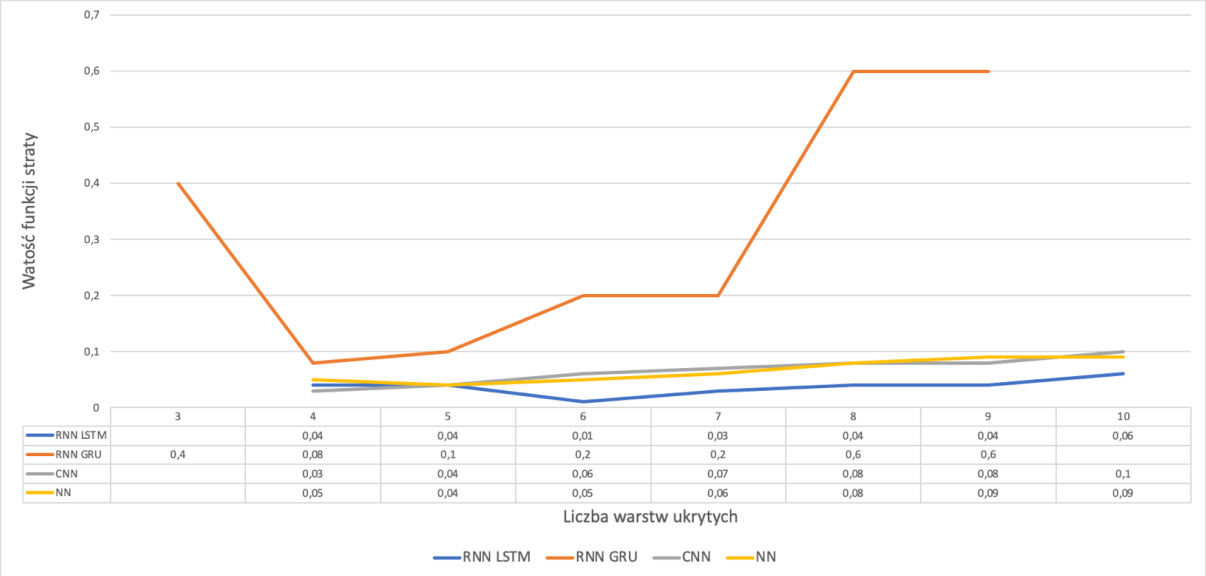
Nazwa parametru	Opis	Wartość
Funkcja jądra	Funkcja, która pozwala na odwzorowanie danych na nowej przestrzeni	rbf
gamma	Parametr funkcji jądra	1/800
C	Parametr pozwalający na normalizację	1

Dostrajając hiperparametry dla algorytmu SVM przeprowadzono eksperymentalne porównanie wyników osiągalnych przez opracowany model z wykorzystaniem różnych funkcji jądra. Najlepsze wyniki osiągnęła Radialna Funkcja Bazowa (ang. Radial Basis Function - RBF). Warto także zauważyć, że rezultaty osiągane przez klasyfikatory oparte o SVM z funkcjami jądra: Sigmoid oraz Funkcją Wielomianową były gorsze od RBF o około 10%.

Tabela 6.5: Wybrane hiperparametry wykorzystywane w algorytmie NN [118]

Nazwa parametru	Opis	Wartość
Liczba warstw	Liczba warstw, z których pierwsza jest warstwą danych wejściowych, ostatnia warstwą danych wyjściowych, a wszystkie pomiędzy nimi to warstwy ukryte	4 dla CNN, 5 dla RNN GRU i NN oraz 6 dla RNN LSTM
Funkcja aktywacji	Funkcja służąca do obliczenia wartości na wyjściu neuronu	ReLU dla warstw ukrytych, Sigmoid dla warstwy wyjściowej
Normalizacja	Algorytm oraz odpowiadające mu współczynniki pozwalające na minimalizację prawdopodobieństwa przetrenowania algorytmu	Ridge Regression, $\alpha(0.1)$

W przypadku Sieci Neuronowych (Tabela 6.5) przeprowadzono analizę wpływu liczby warstw ukrytych na dokładność klasyfikatora uwzględniając czas trwania jednej epoki nauczania. Wykonano badania wpływu liczby warstw ukrytych na wartość Funkcji Straty (reprezentującej umiejętność modelu do nauki), która została przedstawiona na Rys. 6.4. Optymalne wartości dla badanych algorytmów zostały przedstawione w Tabeli 6.5. W przypadku wszystkich weryfikowanych wariantów, zwiększenie liczby warstw ukrytych powodowało wzrost wartości Funkcji Straty (której wartość minimalizujemy).



Rys. 6.4: Zależność wartości Funkcji Straty od liczby warstw ukrytych dla różnych algorytmów

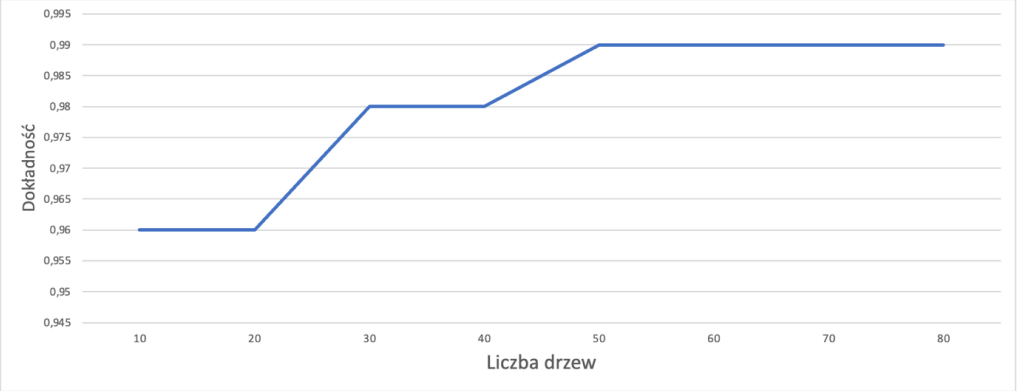
Funkcja aktywacji ReLU charakteryzuję się tym, że tylko pewna liczba neuronów jest aktywowana w tym samym czasie. Dzięki temu istnieje mniejsze prawdopodobieństwo

zaniku gradientu, dlatego funkcja ReLU została wybrana jako funkcja aktywacji dla warstw ukrytych. Jako że problem opisywany w niniejszej rozprawie jest rozwiązywalny przez klasyfikator binarny (wynik działania predykcji dla pojedynczej próbki wynosi 0 jeśli podatność bezpieczeństwa nie jest istotna oraz 1 jeśli wymaga pilnego usunięcia z systemu, w którym została wykryta) wykorzystano dla warstwy wyjściowej funkcję Sigmoid [119].

Tabela 6.6: Wybrane hiperparametry wykorzystywane w algorytmie Lasu Losowego [120]

Nazwa parametru	Opis	Wartość
Liczba drzew	Liczba drzew, które zostaną wygenerowane	50
Kryterium podziału	Funkcja weryfikująca jakość podziału	Gini
Maksymalna głębokość drzewa	Maksymalna liczba węzłów, która może być osiągnięta przez drzewo	Brak
Minimalna liczba przykładów dla konkretnego podziału	Warunek, jaki musi spełnić podział	2

Dostrajając hiperparametry dla algorytmu Lasu Losowego (Tabela 6.6) ustalono liczbę drzew na 50 (większa liczba nie wpływała na dokładność modelu, natomiast mniejsza pozwalała na uzyskiwanie mniej dokładnych wyników). Analiza wpływu liczby drzew na dokładność uzyskiwaną przez opracowany klasyfikator została zilustrowana na Rys. 6.5.



Rys. 6.5: Zależność wartości metryki dokładności od liczby drzew w Lesie Losowym

Walidacja krzyżowa w tym przypadku nie została wykorzystana, ponieważ zestaw narzędzi (wykorzystane biblioteki) zapewniają funkcje, które pozwalają zapewnienie różnorodności podczas podziału zbioru danych na zestaw testowy oraz treningowy.

Opisany na początku tego rozdziału zestaw danych po poddaniu przetwarzaniu wstępnemu został wykorzystany do nauki, a następnie przetestowania modelu, który został zbudowany na podstawie wybranych algorytmy. Przedstawione scenariusze testowe zostały zaimplementowane w aplikacji MixewayVulnAuditor[35]. Następnym krokiem, który

[35] https://github.com/mixeway/mixewayvulnauditor

pozwoli porównać klasyfikatory zbudowane z wykorzystaniem różnych algorytmów pod kątem wykorzystania w procesie analizy podatności bezpieczeństwa jest analiza wyników przedstawiona w dalszej części niniejszej rozprawy.

7. Wyniki badań eksperymentalnych

Badania eksperymentalne przeprowadzono według metodyki i w konfiguracji opisanej w Rozdziale 6 wykorzystując dane przedstawione i uzyskane za pomocą oprogramowania Mixeway (Rozdział 5). W ramach tych prac wykonano porównanie skuteczności wybranych algorytmów uczenia maszynowego. Zadanie polegało na klasyfikacji podatności bezpieczeństwa do jednej z dwóch grup: błędów wymagających poprawy lub błędów nieistotnych.

Jednym ze sposobów na porównanie działania klasyfikatorów może być ograniczenie się do weryfikacji metryki dokładności [121]. Biorąc jednak pod uwagę naturę dziedziny cyberbezpieczeństwa, istotna jest dokładna analiza w szczególności wykryć fałszywie pozytywnych oraz fałszywie negatywnych. Uwzględniając przypadek użycia opisany w Rozdziale 3.4, czyli wykorzystania zbudowanego klasyfikatora jako źródło informacji dotyczącej bezpieczeństwa w procesie CICD duża liczba wykryć fałszywie pozytywnych może wpływać na częste przerywanie łańcucha dostarczania oprogramowania bez konkretnego uzasadnienia. Takie działanie jest nie do zaakceptowania z biznesowego punktu widzenia. Z drugiej strony sytuacja, w której klasyfikator oznacza podatność jako DRNV (nieistotny błąd), podczas gdy powinna ona być traktowana jako CRV (wysoki priorytet) może doprowadzić do poważnego zagrożenia. Z powodów wymienionych powyżej, aby porównać badane algorytmy, zweryfikowane zostaną wartości precyzji oraz czułości, które określają skuteczność klasyfikatora pod względem wykrywania wartości FP oraz FN. Sposób obliczania wszystkich metryk, istotnych z punktu widzenia proponowanego systemu, zamieszczono poniżej:

$$precyzja = \frac{TP}{TP + FP} \tag{12}$$

$$\text{dokładność} = \frac{TP + TN}{TP + FP + TN + FN} \tag{13}$$

$$\text{czułość} = \frac{TP}{TP + FN} \tag{14}$$

$$F1 = 2\frac{precyzja * \text{czułość}}{precyzja + \text{czułość}} \tag{15}$$

Istnieje wiele innych metryk takich jak: swoistość (ang. specificity), odsetek wyników fałszywie negatywnych (ang. false negative rate) czy odsetek wyników fałszywie prawdziwych (ang. false positive rate) [122]. Nie będą one jednak rozważane, biorąc pod uwagę cel, w jakim ma zostać wykorzystany opracowany klasyfikator.

7.1. Badanie skuteczności modeli zbudowanych w oparciu o przedstawione algorytmy [36]

Badania skuteczności modeli przeprowadzono z wykorzystaniem następujących algorytmów uczenia maszynowego: kilku wariantów Sieci Neuronowych (NN, CNN, RNN LSTM oraz RNN GRU), SVM oraz algorytm Lasu Losowego. Wybór konkretnych rozwiązań został dokonany na podstawie wyników badań opublikowanych przez innych badaczy w literaturze światowej dla analogicznych zastosowań [41], [42]. Rezultaty uzyskane w wyniku przeprowadzonych analiz zawarto w Tabeli 7.1. Dokładność oraz precyzja uzyskane przez NN, RNN LSTM, CNN oraz RF znajdują się na bardzo zbliżonym i wysokim poziomie (0,98 - 0,99), podczas gdy wyniki dla tych metryk osiągane przez RNN GRU oraz SVM są gorsze (odpowiednio 0,92 oraz 0,96). W przypadku czułości RNN GRU uzyskał wynik 0,92, który mimo, że jest wysoki (>0,9) jest znacznie gorszy od wyników uzyskiwanych przez pozostałe algorytmy (0,98-0,99). Wartość metryki F1 uzyskana przez badane algorytmy wykazuje podobny charakter do wartości dokładności oraz precyzji, gdzie: NN, RNN LSTM, CNN oraz RF osiągają bardzo wysoką skuteczność (0,98 - 0,99), torchę słabiej wypada SVM (0,96) natomiast najgorzej RNN GRU 0,89 i 0,87. W przypadku czasu nauki modelu najlepszy wynik uzyskał RF - 10 sekund, natomiast dla podstawowa wersja NN wymagała 16 sekund (dla każdej epoki), co jest dużą różnicą w porównaniu do wariantu RNN LSTM potrzebującego 390 sekund na proces nauki modelu.

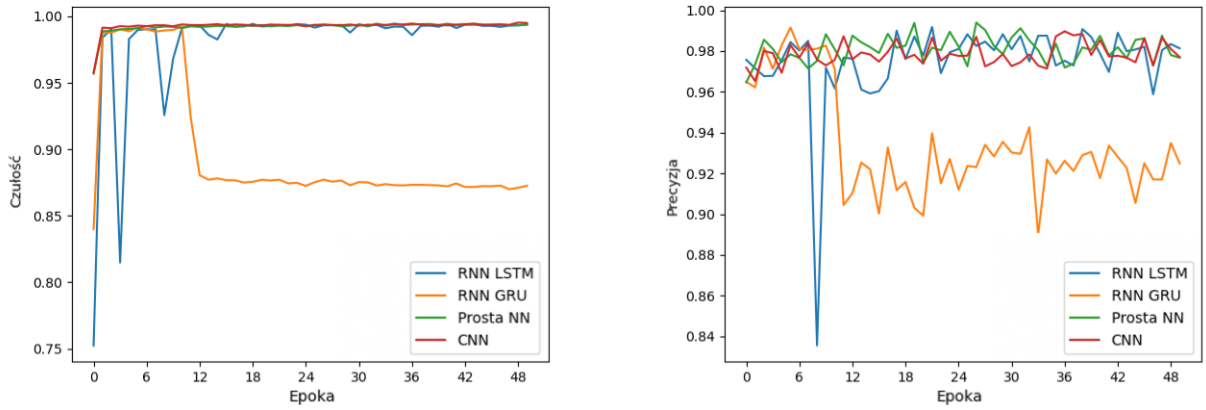
Tabela 7.1: Wyniki porównania skuteczności wybranych algorytmów ML w klasyfikacji podatności bezpieczeństwa

Algorytm	Dokładność	Precyzja	Czułość	F1	Czas nauki [s]
NN	0,98	0,96	0,99	0,98	16
RNN LSTM	0,99	0,98	0,98	0,98	390
RNN GRU	0,92	0,87	0,92	0,89	375
CNN	0,99	0,98	0,99	0,98	27
RF	0,98	0,99	0,98	0,99	10
SVM	0,96	0,97	0,96	0,96	644

Na Rys. 7.1 przedstawiono wykres zależności metryk czułości i precyzji od liczby epok w procesie uczenia modelu. W przypadku algorytmu RNN GRU wyraźnie widać w obu przypadkach znaczny spadek skuteczności modelu po wykonaniu 10 epok. Pozostałe warianty Sieci Neuronowych, po przekroczeniu tej liczby powtórzeń procesu nauczania, zachowują stabilne wartości badanych metryk. Między 10 a 50 epoką warianty CNN, RNN LSTM oraz prosta NN osiągają zbliżone wyniki.

Biorąc pod uwagę wartości, które zostały wykorzystywane w celu obliczenia metryk takich jak precyzja czy czułość (przedstawionych w Tabeli 7.2) można zaobserwować, że

[36] Wyniki prezentowane w tym rozdziale zostały opublikowane w: *Siewruk, G., & Mazurczyk, W. (2021). Context-Aware Software Vulnerability Classification Using Machine Learning. IEEE Access, 9, 88852-88867.*



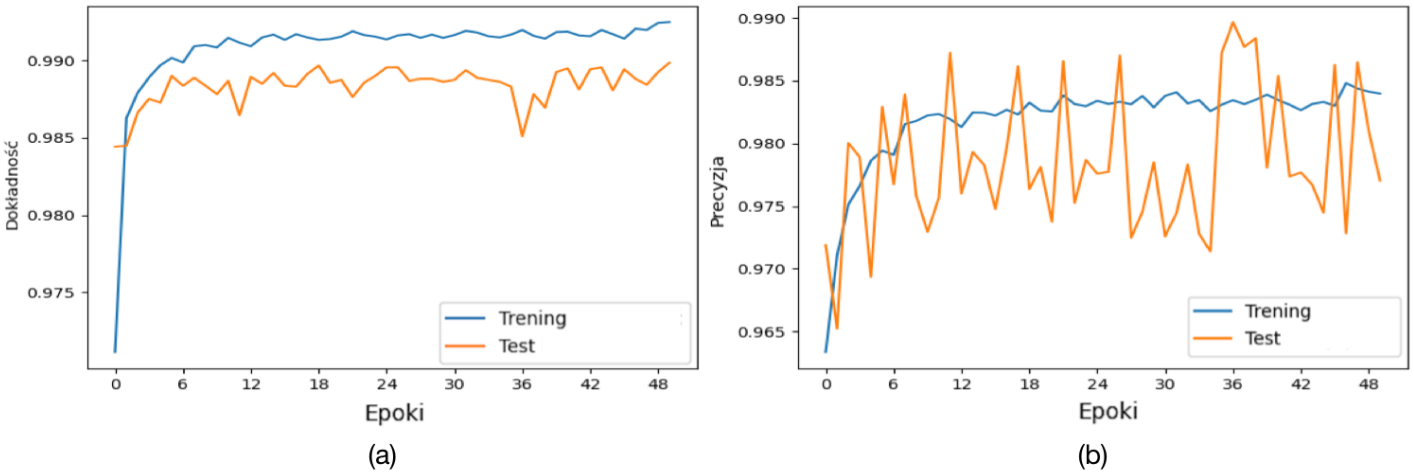
Rys. 7.1: Wartości metryk czułości i precyzji dla różnych wariantów Sieci Neuronowych

Tabela 7.2: Poziom TP, FP, FN i TN dla analizowanych algorytmów uczenia maszynowego

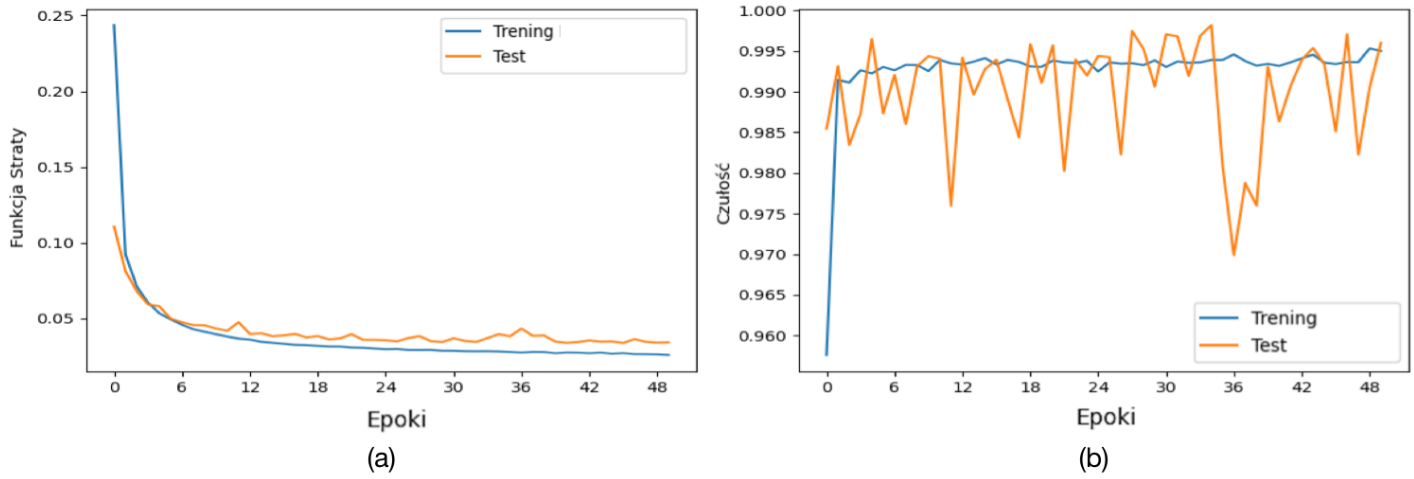
Algorytm	TP [Liczba/%]	FP [Liczba/%]	TN [Liczba/%]	FN [Liczba/%]
NN	5 797 / 35,8	181 / 1,1	10 185 / 63,1	10 / 0,1
RNN LSTM	5 733 / 35,4	72 / 0,4	10 294 / 63,6	74 / 0,5
RNN GRU	5 187 / 32,1	760 / 4,1	9 806 / 60,6	420 / 2,6
CNN	5 763 / 35,6	109 / 0,7	10 257 / 63,4	44 / 0,7
RF	10 321 / 63,8	85 / 0,5	5 658 / 34,9	109 / 0,7
SVM	10 069 / 62,3	372 / 2,3	5 573 / 34,4	159 / 0,9

algorytmy RNN LSTM oraz NN uzyskały najmniejszą liczba próbek oznaczonych jako FN. Jednakże, aby możliwe było wykorzystanie weryfikowanego klasyfikatora w procesie CICD, pod uwagę muszą być brane zarówno poziom FP jak i FN. Algorytm SVM oraz RNN LSTM z liczbą FP na poziomie 372 (dla SVM) oraz 760 (dla RN LSTM) jest zdecydowanie mniej efektywny niż RNN LSTM, który oznaczył zaledwie 72 próbki wartościami, które były FP.

Wyniki dla algorytmów SVM oraz RF przedstawione są w Tabeli 7.1. Prezentowane wartości są uśrednionymi metrykami uzyskanymi podczas dwudziestokrotnego uruchomienia procesu nauczania oraz testowania modelu. Różnice pomiędzy konkretnymi uruchomieniami całego procesu wahają się w granicy 1% przez co można uznać, że wartości metryk uzyskanych przez te klasyfikatory są stabilne.

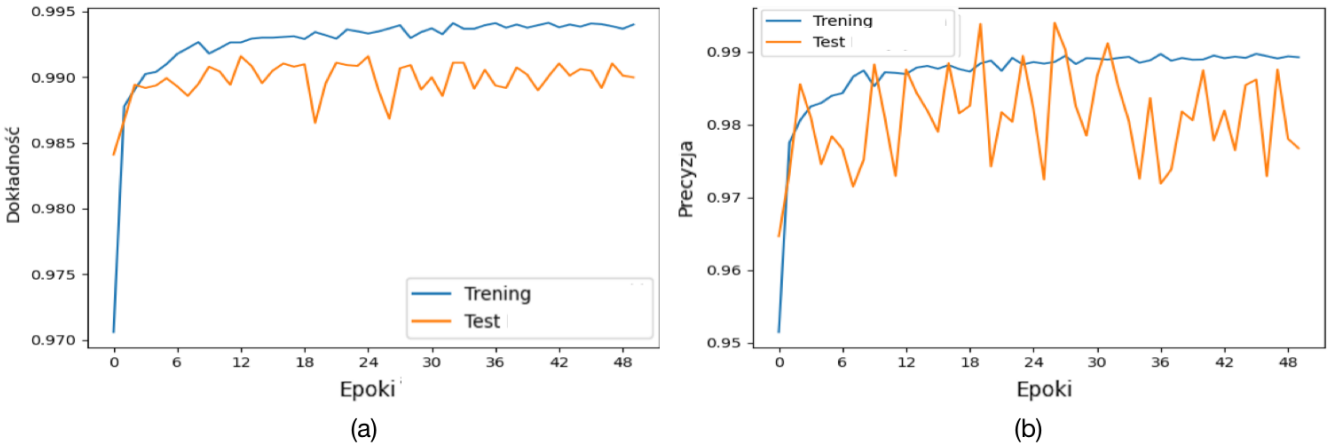


Rys. 7.2: Wartości metryk uzyskane przez algorytm CNN: (a) Dokładność, (b) Precyzja

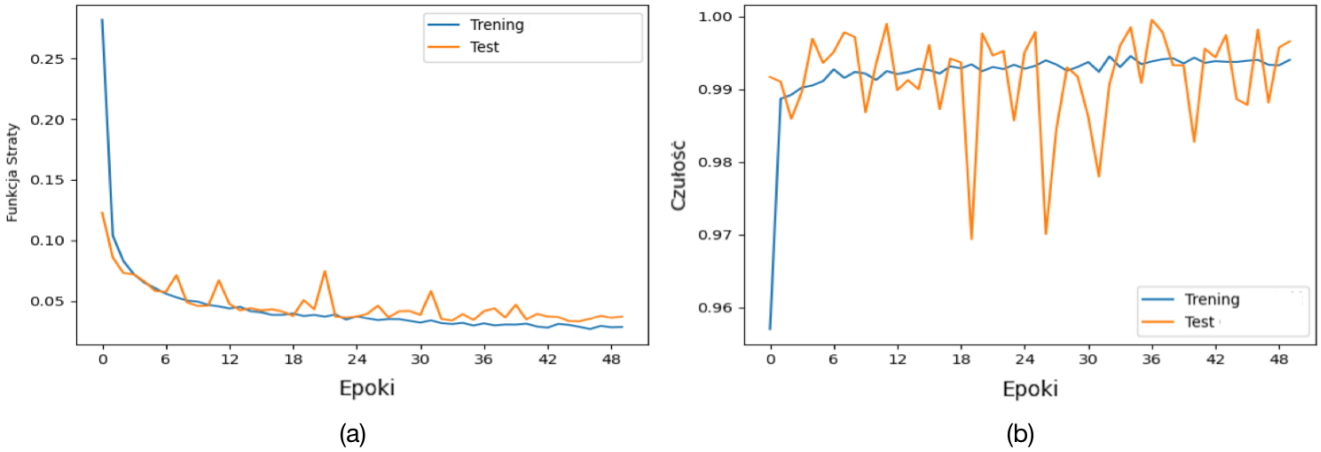


Rys. 7.3: Wartości metryk uzyskane przez algorytm CNN: (a) Funkcja Straty, (b) Czułość

Rys. 7.2 przedstawia wartości metryk dokładności oraz precyzji uzyskane przez algorytm CNN w zależności od epoki. Można zaobserwować, że w przypadku obydwu metryk algorytm uzyskuje stabilny oraz wysoki wynik po osiągnięciu 10 przebiegów. Na Rys. 7.3 przedstawiono zależność wartości funkcji straty oraz czułości od epoki dla tego samego klasyfikatora. Wartość funkcji straty osiąga wynik 0,05 już dla 6 epoki co oznacza, że model dla zadanego zestawu danych był w stanie obliczyć odpowiednie wagi bardzo szybko. W przypadku metryki czułości optymalny wynik został uzyskany jeszcze szybciej - już po 3 przebiegu.

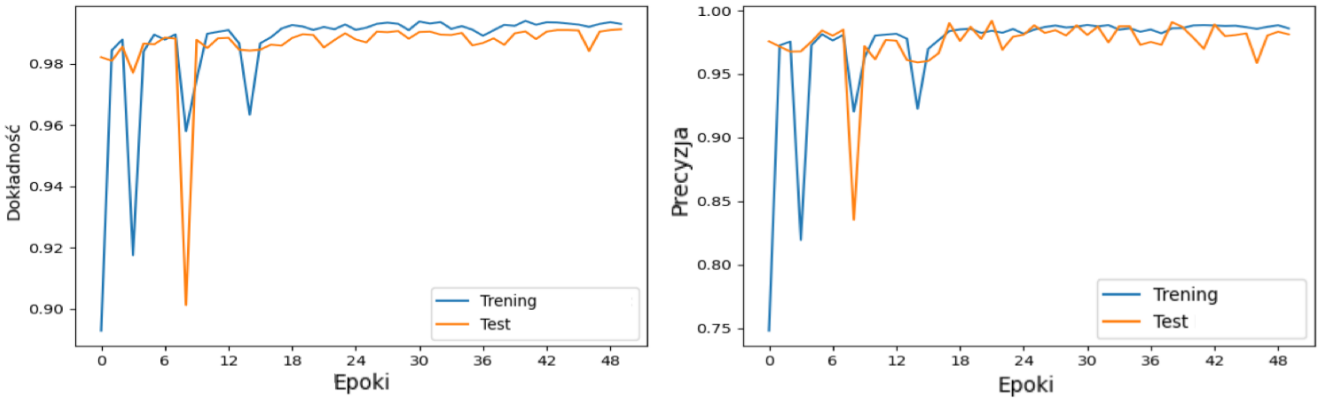


Rys. 7.4: Wartości metryk uzyskanych przez algorytm NN (prosta wersja): (a) Dokładność, (b) Precyzja



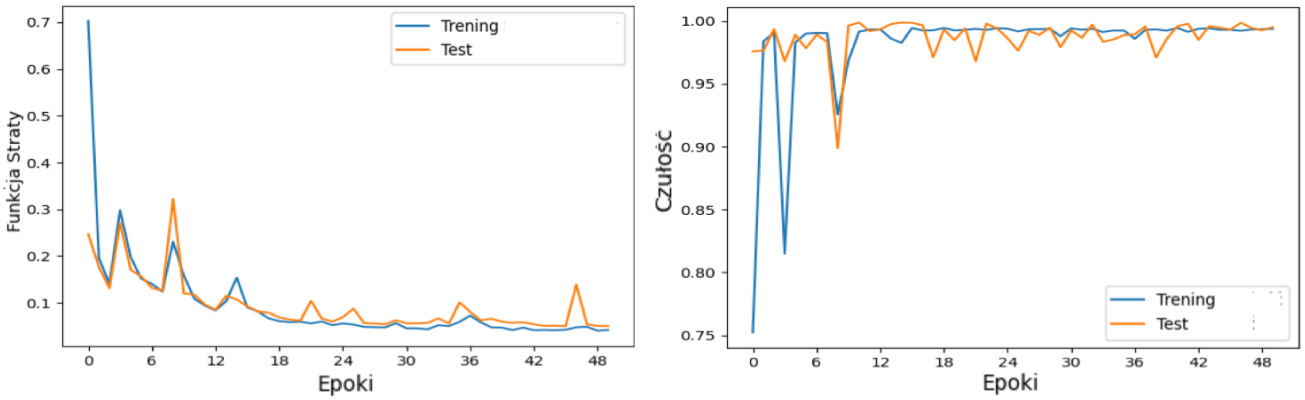
Rys. 7.5: Wartości metryk uzyskanych przez algorytm NN (prosta wersja): (a) Funkcja Straty, (b) Czułość

Rys. 7.4 przedstawia wartość metryk dokładności oraz precyzji, które osiągają wartość 0,99 po upływie sześciu epok dla prostej wersji Sieci Neuronowej. Na Rys. 7.5 obrazuje wartość funkcji straty oraz czułości w zależności od przebiegu procesu nauczania. W przypadku tego algorytmu proces nauczania osiągnął optymalne wyniki po upływie 6 epok (wartość funkcji straty osiągnęła wartość 0,05) natomiast czułość osiągnęła wysoki wynik po upływie 3 epok.



(a) (b)

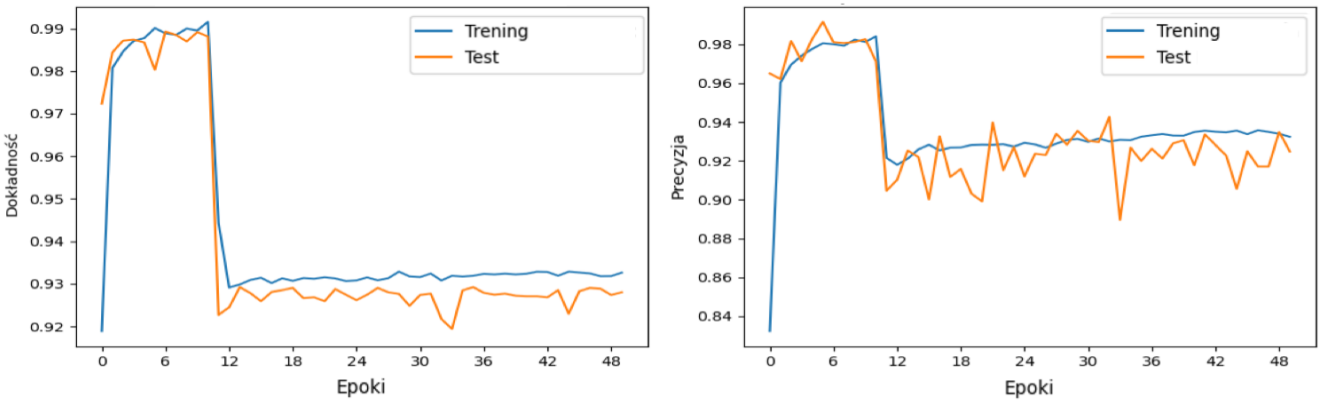
Rys. 7.6: Wartości metryk uzyskanych przez algorytm RNN LTSM: (a) Dokładność, (b) Precyzja



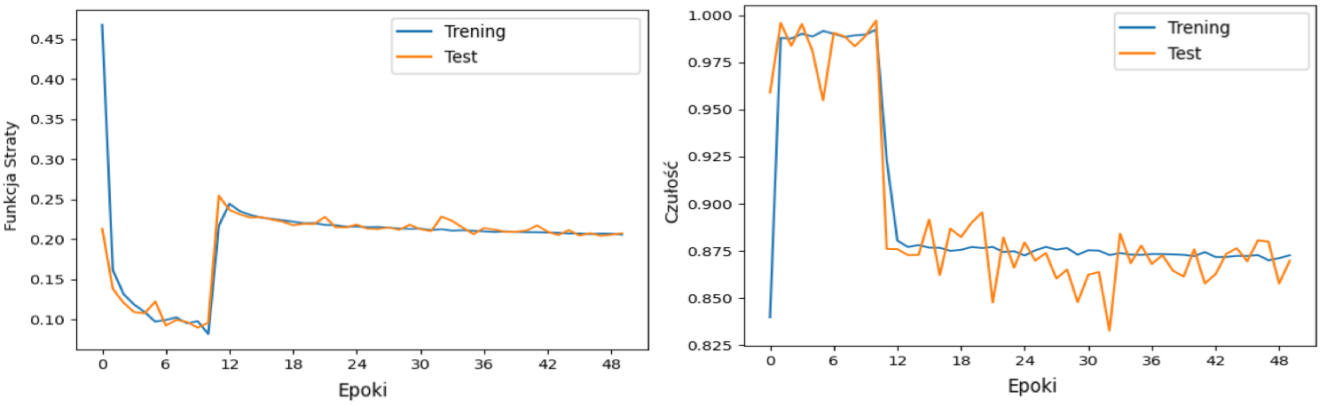
(a) (b)

Rys. 7.7: Wartości metryk uzyskanych przez algorytm RNN LTSM: (a) Funkcja Straty, (b) Czułość

Rys. 7.6 przedstawia wartość metryk dokładności oraz precyzji w zależności od liczby epok dla algorytmu RNN LSTM. Dokładność oraz precyzja osiąga wynik 0,98 po upływie 10 epok. Natomiast na Rys. 7.7 zobrazowano zależność wartości funkcji straty, oraz czułości od liczby przebiegów procesu nauczania. Poprawne wartości wag dla neuronów zostały osiągnięte po upływie 12 epok (wartość funkcji straty na poziomie 0,1) podczas gdy czułość utrzymywała się na wysokim poziomie już po upływie 3 epok. Warto zwrócić uwagę na stabilność wyników dla wszystkich badanych metryk. Po upływie epoki, w której został osiągnięty optymalny wynik wartości względem siebie nie podlegają większym wahaniom.



Rys. 7.8: Wartości metryk uzyskanych przez algorytm RNN GRU: (a) Dokładność, (b) Precyzja



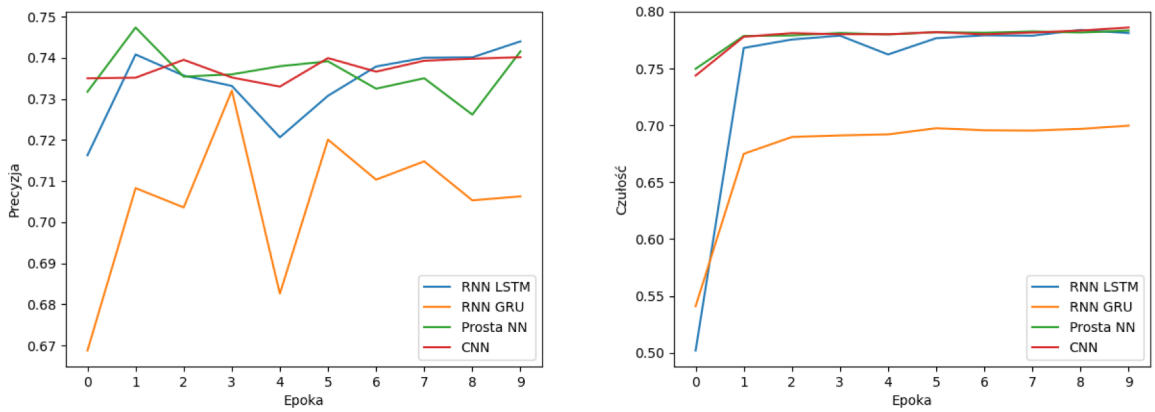
Rys. 7.9: Wartości metryk uzyskanych przez algorytm RNN GRU: (a) Funkcja Straty, (b) Czułość

Rys. 7.8 przedstawia wartości metryk dokładności oraz precyzji natomiast Rys. 7.9 wartość funkcji straty oraz czułość dla algorytmu RNN GRU. Opracowany klasyfikator uzyskał optymalne wyniki dla wszystkich badanych metryk po 12 epokach. W następnych przebiegach algorytm w każdej weryfikowanej metryce notuje znaczne spadki. W przypadku dokładności i precyzji wartości spadają o 7-8 punktów procentowych, czułości - o 10 punktów procentowych natomiast w wartość funkcji straty wzrasta niemal 2,5 krotnie.

W przypadku wszystkich wymienionych algorytmów różnice między metrykami dla danych testowych a treningowymi oscylują w granicach 2%. Wynik taki pozwala ocenić, że nie doszło do przetrenowania modelu, a rezultaty prezentowane przez opracowane klasyfikatory są miarodajne i wiarygodne. W przypadku wariantu Sieci Neuronowej RNN LSTM (Rys. 7.6 i 7.7) zaobserwowano powtarzalność zmian w obserwowanych metrykach (średnio co 4-5 epok). Z kolei rezultaty osiągnięte przez RNN GRU zdecydowanie odbiegają charakterem od pozostałych wariantów Sieci Neuronowych (Rys. 7.8 i 7.9). W tym przypadku widać wyraźną różnicę po osiągnięciu granicy 10 epok, po których wyniki uzyskiwane przez klasyfikator są znacząco gorsze od pozostałych badanych wariantów.

Biorąc od uwagę wyniki, które zostały opisane powyżej można ocenić, że problem klasyfikacji podatności bezpieczeństwa jest rozwiązywany z dużym sukcesem przez modele opracowane z wykorzystaniem algorytmów: Lasu Losowego, Rekurencyjnej Sieci Neuronowej z komórkami pamięci typu LSTM oraz Konwolucyjnej Sieci Neuronowej.

Aby potwierdzić poprawność opracowanego modelu oraz zweryfikować wpływ, jaki może mieć niepoprawnie przygotowany zestaw danych służący do procesu trenowania na osiągane wyniki, przygotowany został także zestaw danych, który został poddany manipulacji. Wartości oceny zostały zmienione dla 30% podatności, które znajdowały się w zestawie danych treningowych [123]. W wyniku tej operacji zestawy danych, które zostały wykorzystane w celu nauki modelu, mogą zawierać sprzeczności, przez co taka sama podatność bezpieczeństwa znaleziona w identycznym kontekście wykorzystania aplikacji może przyjmować różne wartości oceny. Taka sytuacja nigdy nie powinna mieć miejsca. Wartości metryk dla klasyfikatorów zbudowanych na podstawie procesu uczenia z wykorzystaniem tego typu danych powinny zdecydowanie odbiegać od wartości, które zostały przedstawione powyżej.



Rys. 7.10: Wartości metryk precyzji oraz czułości w zależności od epoki dla modelu zbudowanego na podstawie zmanipulowanych danych

Na Rys. 7.10 przedstawiono wyniki uzyskane przez modele zbudowane na podstawie badanych algorytmów oraz zmanipulowanych danych. Wyraźnie widoczny jest spadek

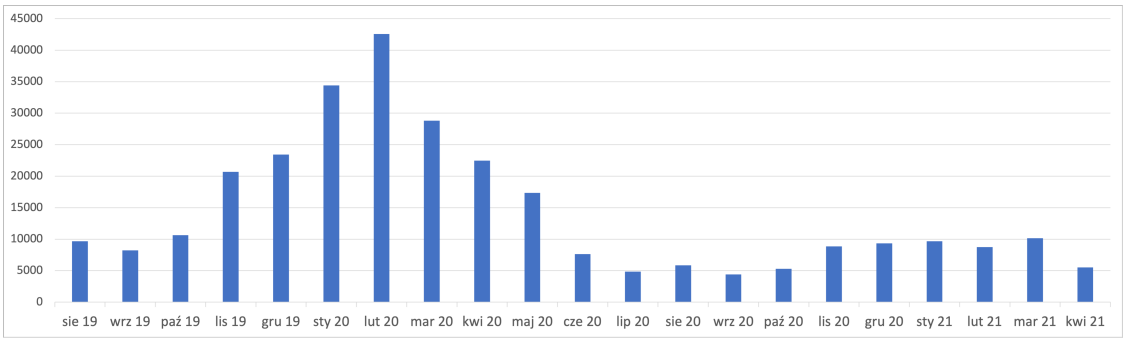
wartości metryk takich jak precyzja oraz czułość. Spadek wymienionych metryk jest zauważalny dla każdego badanego algorytmu (w porównaniu do wyników, które zostały przedstawione powyżej).Czułość dla algorytmów RNN LSTM, NN oraz CNN jest na porównywalnym poziomie - 0,77 podczas, gdy RNN GRU osiągnął wynik 0,69. Rezultaty te są zdecydowanie gorsze na tle osiąganych wyników przez modele zbudowane z w oparciu o poprawne dane - odpowiednio 0,97 i 0,92 (Tabela 7.1). Podobna obserwacja dotyczy metryki precyzji. W przypadku algorytmu RF wynik precyzji wyniósł 0,9 przy 0,75 czułości natomiast SVM uzyskał 0,66 precyzji przy 0.84 czułości. Wpływ manipulacji danych na rezultaty osiągane przez badane klasyfikatory jest zauważalny. Zarówno metryka czułości jak i precyzji spada średnio o 0,2 przy 30% rekordów, których klasyfikacja została zmieniona.

Biorąc pod uwagę wszystkie zaprezentowane wyniki można jednoznacznie ocenić, iż możliwe jest stworzenie mechanizmu, który w automatyczny sposób będzie klasyfikował podatności bezpieczeństwa wykryte przez różnego rodzaju oprogramowanie skanujące aplikację. Wysokie wartości metryk takich jak dokładność, precyzja i czułość, które wynikają z niewielkiej liczby fałszywych alarmów (Tabela 7.2) pozwalają na zbudowanie procesu decyzyjnego, który w inteligenty sposób będzie w stanie ocenić, czy aplikacja, w której wdrażana jest zmiana jest bezpieczna (czy spełnia przyjęte w organizacji polityki bezpieczeństwa) czy nie. Implementacja tak opracowanego mechanizmu w oprogramowaniu Mixeway, które w z natury w prosty i przyjazny sposób integruje się z mechanizmami CICD pozwala na zaproponowanie kompromisu między bezpieczeństwem a szybkością wdrożenia zmiany w aplikacji, który bez dostępu do Mixeway jest niemożliwy do osiągnięcia.

7.2. Wyniki uzyskane dzięki wdrożeniu systemu Mixeway w infrastrukturze operatora telekomunikacyjnego

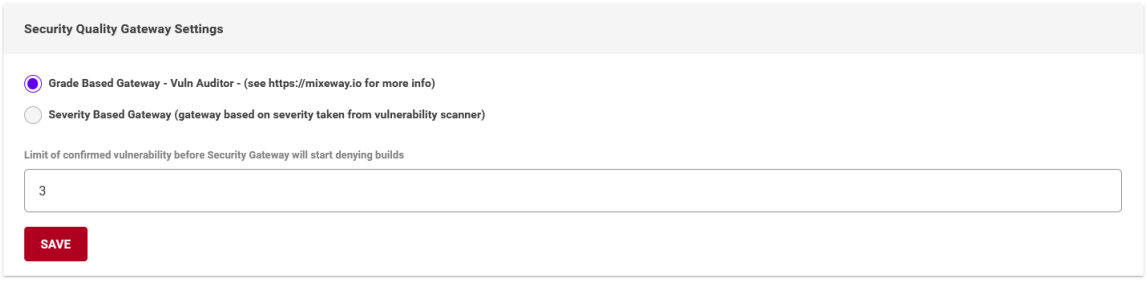
Jak już wspomniano, oprogramowanie Mixeway zostało wdrożone w infrastrukturze operatora telekomunikacyjnego w kilku etapach (Rozdział 5). Dzięki uruchomieniu modułu odpowiedzialnego za realizację testów podatności możliwa była orkiestracja skanerów typu: SAST, DAST oraz OpenSource. Wyniki uzyskane przez ten moduł zaprezentowano na Rys. 7.11.

Średnia miesięczna liczba wykrytych podatności rośnie do wartości około 42 000 w lutym 2020. Wzrost wynika z regularnego obejmowania automatycznymi skanami podatności coraz większej liczby systemów teleinformatycznych w infrastrukturze operatora telekomunikacyjnego. Każda wykryta podatność w opisywanym procesie jest raportowana do odpowiedniej jednostki w ustalonym trybie. Po objęciu automatycznymi testami wszystkich aplikacji rozwijanych w trybie zwinnym widoczny jest wyraźny trend spadkowy. Usunięcie 87% podatności (porównanie lutego 2020 i kwietnia 2021) potwierdza skuteczność wdrożonego rozwiązania.



Rys. 7.11: Średnia liczba podatności wykryta we wszystkich systemach w danym miesiącu

Wyniki działania badanych algorytmów przedstawione w Rozdziale 7.1 udowadniają efektywność opracowanego modelu, który klasyfikuje podatności bezpieczeństwa wykryte przez dowolny skaner podatności, w wyniku czego każdy błąd jest oznaczany jako CRV oraz DRNV. Aby zweryfikować czy opracowany klasyfikator można wykorzystać w łańcuchu dostarczania oprogramowania wykorzystano oprogramowanie Mixeway (przedstawione w Rozdziale 5). W badaniach wykorzystano model zbudowany z wykorzystaniem algorytmu RNN LSTM. Zaimplementowana bramka bezpieczeństwa jest możliwa do skonfigurowania. Rys. 7.12 przedstawia konfigurację ustawienia, przy pomocy której definiowana jest maksymalna liczba podatności oznaczonych jako CRV, aby zmiana w weryfikowanej aplikacji została uznana jako bezpieczna.

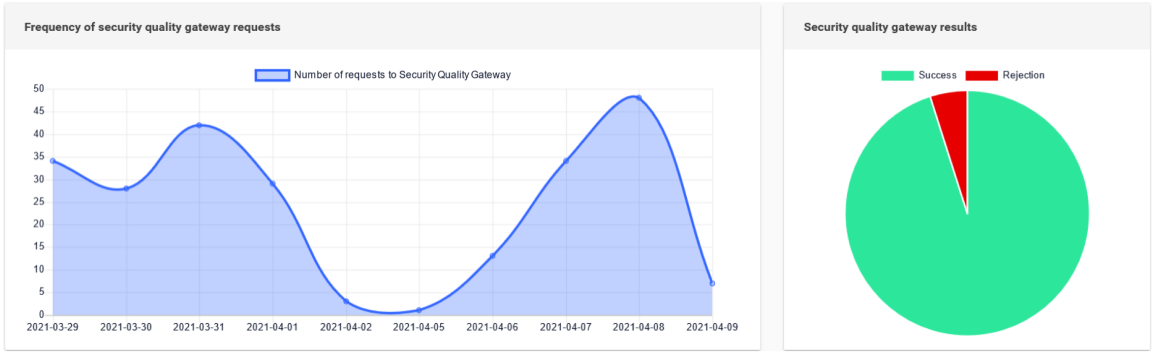


Rys. 7.12: Konfiguracja bramki bezpieczeństwa w oprogramowaniu Mixeway z wykorzystaniem aplikacji VulnAuditor, która implementuje opracowany model

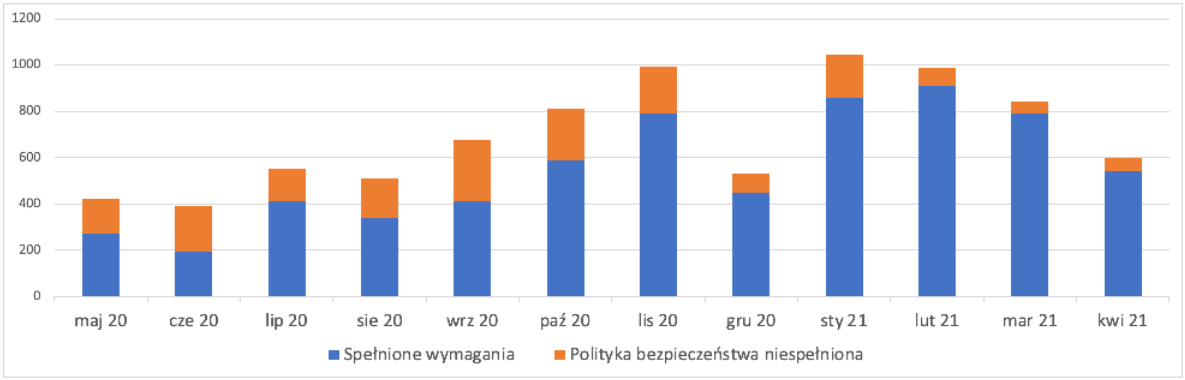
Konfiguracja przedstawiona na Rys. 7.12, zapewnia, że jeśli w aplikacji zostanie wykrytych więcej niż 3 podatności bezpieczeństwa sklasyfikowane jako CRV łańcuch dostarczenia oprogramowania zostanie przerwany.

Tak opracowana bramka bezpieczeństwa została umieszczona w procesie CICD dla 130 aplikacji, a dane na temat jej wyników były zbierane przez 9 miesięcy (tj. w okresie od 07.2020 do 04.2021).

Rys. 7.13 przedstawia informacje dotyczące sposobu działania wdrożonego mechanizmu. Widać, że średnio w ciągu tygodnia uruchomionych jest 20 zmian w kodzie źródłowym różnych aplikacji. Stosunek odrzuceń wdrożenia aplikacji wynosi 5% co znaczy, że co 20 wdrożenie zawiera podatności, które oznaczane są jako konieczne do poprawy.



Rys. 7.13: Liczba odpytań bramki bezpieczeństwa w stosunku dziennym (lewy wykres) oraz stosunek informacji o akceptacji i odrzuceniu zmiany ze względu na bezpieczeństwo (prawy wykres)



Rys. 7.14: Miesięczna liczba weryfikacji bramki bezpieczeństwa z podziałem na spełnienie polityki bezpieczeństwa oraz jej złamanie

Tabela 7.3: Miesięczna liczba weryfikacji bramki bezpieczeństwa oraz stosunek wdrożeń niespełniających wymagań bezpieczeństwa

Miesiąc	Spełnione wymagania	Polityka bezpieczeństwa niespełniona	% Wdrożeń niespełniających polityki bezpieczeństwa
maj 20	273	150	35%
cze 20	192	198	51%
lip 20	412	142	26%
sie 20	340	171	33%
wrz 20	412	266	39%
paź 20	590	222	27%
lis 20	790	201	20%
gru 20	450	80	15%
sty 21	859	187	18%
lut 21	910	77	8%
mar 21	789	55	7%
kwi 21	540	56	9%

Warto zauważyć, że podczas pierwszych tygodni działania wdrożonego oprogramowania odsetek ten wynosił między 30% a 35% (por. Rys. 7.14, Tabela 7.3). Spadek tych wartości do 9% (czyli ponad 3-4 krotny) oznacza, że:

- Błędy zgłoszone do poprawy przez zespoły programistyczne zostały usunięte.
- Błędy, które zostały zidentyfikowane jako takie, które nie są istotne (a były oznaczone przez klasyfikator jako CRV) zasiliły model ponownie, w wyniku czego w przy kolejnych uruchomieniach były klasyfikowane poprawnie (zgodnie z kontekstem wykorzystywania aplikacji)
- Czas realizacji testów podatności uległ znacznemu skróceniu. Po wdrożeniu Mixeway, wszystkie operacje, inicjujące skany oraz weryfikujące wyniki są realizowane w sposób automatyczny

Biorąc pod uwagę zarówno wyniki, które zostały uzyskane przez opracowany klasyfikator jak i rezultaty wdrożenia Mixeway w infrastrukturze operatora telekomunikacyjnego należy stwierdzić, że przedstawione wdrożenie jest udane. Zysk z przeprowadzonych badań oraz z zrealizowanego rozwiązania zarówno dla organizacji jak i dziedziny nauki, do której należy cyberbezpieczeństwo oraz uczenie maszynowe jest znaczący. Rozwiązanie polegające na jednoznacznym określeniu czy podatność bezpieczeństwa wykryta przez narzędzia automatyczne wymaga poprawy czy nie, w takiej formie oraz skali (niezależne od źródła, z którego pochodzi informacja o wykrytym błędzie) nie było wcześniej prezentowane. Dzięki możliwości wykorzystania Mixeway w procesach automatycznego dostarczania oprogramowania możliwe jest zarówno zachowanie bezpieczeństwa teleinformatycznego jak i odpowiedniej zwinności zespołów programistycznych pozwalającej na skróceniu czasu dostarczania nowych wersji oprogramowania.

8. Podsumowanie

Problem omawiany w niniejszej rozprawie dotyczy możliwości wykorzystania klasyfikacji podatności bezpieczeństwa w łańcuchu dostarczania oprogramowania. Cele, które zostały określone na początku pracy to: porównanie dokładności klasyfikatorów podatności bezpieczeństwa, opracowanie rozwiązania implementującego opracowany klasyfikator, wdrożenie zaprojektowanego systemu w infrastrukturze operatora telekomunikacyjnego raz udowodnienie skuteczności wdrożonego oprogramowania bazując na danych historycznych.

W wyniku realizowanych prac powstał system Mixeway. Zadania realizowane przez opracowane rozwiązanie to: orkiestracja narzędzi bezpieczeństwa, klasyfikacja wykrytych podatności oraz udostępnienie funkcjonalności bramki bezpieczeństwa, która weryfikuje czy konkretna aplikacja spełnia politykę bezpieczeństwa przyjętą w organizacji.

Rezultaty przeprowadzonych w ramach rozprawy badań zostały opublikowane w następujących pracach naukowych:

- Siewruk G, Mazurczyk W. Context-Aware Software Vulnerability Classification using Machine Learning. IEEE Access. 2021 Apr 23.
- Siewruk G, Mazurczyk W, Karpiński A. Security Assurance in DevOps Methodologies and Related Environments. International Journal of Electronics and Telecommunications. 2019 May 19;65(2):211-6.

oraz zaprezentowane na konferencjach branżowych:

- Siewruk G. O tym jak (nie)łatwo dodać Sec do Dev(Sec)Ops w świecie telco. PLNOG19. Kraków 2019.
- Siewruk G. O tym jak mało jest Sec w Dev(Sec)Ops podczas testów aplikacji. WTH19. Warszawa 2019.
- Siewruk G. Klasyfikacja podatności jako weryfikacja w procesach automatycznego dostarczania oprogramowania. Advanced Threat Summit. Warszawa 2020.

Dodatkowo, wyniki uzyskane przez Mixeway po wdrożeniu w Orange Polska zostały docenione poprzez przyznanie następujących nagród: Nagroda szefa funkcji Sieć i Technologie (przyznane przez Orange Polska), Security & Privacy Awards (przyznane na poziomie międzynarodowej Grupy Orange). Prace nad rozwiązaniem zostały również opisane w artykule "Orange Funds New CI/CD Security Tool" opublikowanym przez *HardenStance Briefing*[37]. Mixeway w momencie pisania niniejszej rozprawy został pobrany ponad 15 000 za pośrednictwem portali DockerHub oraz GitHub. Zgłaszane przez internautów prośby o nowe funkcjonalności potwierdzają zainteresowanie dalszym rozwojem nad opracowaną aplikacją przez badaczy pochodzących z różnych branż.

Tezy sformułowane na początku niniejszej rozprawy to:

- Rekurencyjne sieci neuronowe z uwagi na charakter działania są algorytmem, który

[37] https://www.hardenstance.com/wp-content/uploads/2020/04/Orange-Funds-New-CICD-Security-Tool-FINAL.pdf

jest odpowiedni do budowy klasyfikatora podatności bezpieczeństwa z wykorzystaniem technik przetwarzania języka naturalnego.

- Wdrożenie systemu teleinformatycznego, który pełni rolę orkiestratora narzędzi bezpieczeństwa oraz wykorzystuje sieci neuronowe w celu określenia, które z wyrytych podatności są konieczne do poprawy przez zespoły programistyczne, znacząco wpływa na liczbę przeprowadzonych testów bezpieczeństwa oraz poziom zabezpieczeń projektu, w ramach którego zostanie uruchomiony.

W przypadku pierwszej tezy wyniki przedstawione w Rozdziale 7.1 udowadniają skuteczność modeli zbudowanych z wykorzystaniem badanych algorytmów. Rekurencyjna sieć neuronowa z komórkami pamięci typu LSTM uzyskała wyniki metryk precyzji i czułości na poziomie 98%-99%, charakteryzując się niewielkim odsetkiem podatności bezpieczeństwa błędnie sklasyfikowanych (FP oraz FN). Na podstawie tych wyników można jednoznacznie ocenić, że model zbudowany na podstawie rekurencyjnych sieci neuronowych osiąga zadowalające wyniki dla klasyfikacji podatności bezpieczeństwa.

Biorąc pod uwagę liczbę testów podatności wykonywanych w infrastrukturze operatora telekomunikacyjnego przed i po wdrożeniu oprogramowania Mixeway (por. Rozdziały 5.4 oraz 5.5) można zauważyć znaczące różnice. W każdej kategorii testów wzrost liczby wykonań jest znaczący. W przypadku testów typu SAST oraz DAST wdrożenie omawianego rozwiązania spowodowało ponad 30-krotny wzrost liczby wykonywanych skanowań. Wdrożenie Mixeway pozwoliło na realizację testów weryfikujących bezpieczeństwo bibliotek o otwartym kodzie źródłowym, co przed wdrożeniem nie było możliwe do zrealizowania (biorąc pod uwagę centralne raportowanie). Aby potwierdzić wpływ opracowanego oprogramowania na poziom zabezpieczeń weryfikowanych projektów, warto przypomnieć o różnicy w liczbie aplikacji, które spełniały politykę bezpieczeństwa na początku działania systemu oraz po 9 miesiącach (por. Rozdział 7.2). Dzięki mechanizmom zaimplementowanym w oprogramowaniu Mixeway możliwe było zmniejszenie odsetka wdrożeń niespełniających polityki bezpieczeństwa (posiadających podatności wymagające poprawy) z 35% do 5%. Tak znacząca zmiana pozwala obronić tezę postawioną w niniejszej rozprawie dotyczącą zwiększenia poziomu zabezpieczeń wytwarzanego w organizacji oprogramowania.

8.1. Klasyfikacja podatności z wykorzystaniem algorytmów uczenia maszynowego

Podczas przeprowadzonych badań zweryfikowano wydajność i efektywność różnych algorytmów uczenia maszynowego w celu zbudowania klasyfikatora podatności bezpieczeństwa. Zestaw danych wykorzystanych podczas eksperymentu zawierał informacje dotyczące podatności (nazwa oraz opis) i aplikacji (kontekst działania oraz wykorzystania), w której błąd został wykryty. Duża część danych, które zawarte są w przygotowanej bazie danych ma charakter opisowy. Aby umożliwić klasyfikatorowi poprawne skorzystanie z tego typu informacji wykorzystano techniki NLP.

Uzyskane wyniki potwierdziły to, że badane algorytmy uzyskują zadowalające wyniki. Metryki precyzji w granicach 98% uzyskane przez rekurencyjne sieci neuronowe oraz las losowy zapewniają odpowiednią skuteczność mechanizmu, który można wykorzystać w celu zabezpieczania procesu CICD.

Należy jednak pamiętać, że przeprowadzone badania eksperymentalne były oparte na zestawie danych pochodzących ze skanów bezpieczeństwa wykonanych dla 130 aplikacji, które w większości przeznaczone są do realizacji zadań charakterystycznych dla obsługi usług operatora telekomunikacyjnego. Z uwagi na fakt wrażliwości przetwarzanych informacji (nie istnieją publicznie dostępne bazy danych zawierające informacje o podatnościach wykrytych w konkretnej działającej aplikacji o znanym kontekście) istnieje duża szansa, że opracowany model może wykazywać się mniejszą skutecznością podczas klasyfikacji podatności pochodzących z aplikacji działających na przykład w infrastrukturze banku. Jednakże, zakładając dostępność błędów bezpieczeństwa w zestawie danych treningowych, które pochodzą z aplikacji działających w kontekście usług operatora telekomunikacyjnego, banku oraz firmy ubezpieczeniowej, oczekiwane jest utrzymanie wysokiej dokładności modelu opartego o rekurencyjne sieci neuronowe korzystające z komórek pamięci typu LSTM. Założenie takie oparte jest na różnicy w konfiguracji algorytmu sieci neuronowych, w którym istnieje bardzo duża liczba hiperparametrów pozwalających na konfigurację modelu w porównaniu do pozostałych algorytmów [124].

8.2. Wykorzystanie opracowanego modelu w realizacji zadań operacyjnych

Omawiane rozwiązanie zostało wdrożone w sieci operatora telekomunikacyjnego oraz włączone do łańcucha dostarczania oprogramowania dla 130 aplikacji. Przez pierwsze tygodnie, w których działała bramka bezpieczeństwa weryfikująca zgodność z polityką zabezpieczeń z wykorzystaniem opracowanego modelu blokowanych było 35% wdrożeń (oznaczanych jako takie, które posiadają istotne podatności bezpieczeństwa wymagające szybkiej poprawy). W wyniku zgłaszanych błędów do zespołów programistycznych i informacji zwrotnej dotyczącej wykryć fałszywie pozytywnych, które zostały przekazane do opracowanego modelu ML w celu ponownej nauki, udało zmniejszyć się liczbę niebezpiecznych wdrożeń do 5%. Przedstawione statystyki pozwalają potwierdzić, że wdrożone oprogramowanie w znaczącym stopniu wpływa na poziom bezpieczeństwa organizacji.

8.3. Kierunki dalszych prac

Istnieje wiele potencjalnych kierunków, w których można rozwijać zaproponowane i zrealizowane w niniejszej rozprawie rozwiązanie. W zakresie opracowania modelu uczenia maszynowego dalsze prace powinny skupić się na pozyskaniu większego zestawu danych, posiadającego podatności wykryte w bardziej zróżnicowanym kontekście. Jednym z kierunków jest analiza możliwości wykorzystania rozwiązania Hackerone [125], które zawiera raporty z testów bezpieczeństwa różnego rodzaju serwisów internetowych wykonywa-

nych przez badaczy bezpieczeństwa. Przygotowanie zestawu danych treningowych, który zawierałby wyniki skanów podatności oraz wyniki testów bezpieczeństwa realizowanych ręcznie przez ekspertów pozwoli na opracowanie bardziej elastycznego modelu.

W kontekście oprogramowania Mixeway dalsze prace powinny skupić się na poprawieniu synergii między tym oprogramowaniem a zespołami programistycznymi. Obecny proces przetwarzania informacji wstecznej, czyli takiej, w której programista informuje, że zgłoszona do poprawy podatność nie jest błędem i nie jest możliwa do wykorzystania przez cyberprzestępcę, powinien zostać poprawiony. Zaimplementowane rozwiązanie zakłada przetwarzanie tego typu informacji w sposób ręczny (ktoś musi zgłosić wystąpienie wartości fałszywie pozytywnej, a ktoś inny musi uruchomić proces ponownego uczenia modelu na podstawie dostarczonych danych). Zautomatyzowanie tego typu operacji wpłynie bardzo pozytywnie na współpracę w tym zakresie między zespołami odpowiedzialnymi za dostarczenie oprogramowania a zespołami bezpieczeństwa. Realizacja tego zadania będzie wymagała dodatkowych integracji m.in. z rozwiązaniami służącymi do zgłaszania problemów lub komunikatorami służącymi do wymiany informacji w zespołach programistycznych. Powinna również zakładać dodatkową warstwę weryfikacji (podatność oznaczona jako FP, powinna być zweryfikowana przez eksperta bezpieczeństwa przed przekazaniem do ponownego procesu nauczania). Funkcjonalność tego typu pozwoli jednak znacznie poprawić proces obsługi błędów bezpieczeństwa, które są zgłaszane do zespołów odpowiedzialnych za rozwój oprogramowania.

Bibliografia

[1] S. Jha, S. Sural, V. Atluri i J. Vaidya, „Specification and Verification of Separation of Duty Constraints in Attribute-Based Access Control”, *IEEE Transactions on Information Forensics and Security*, t. 13, nr. 4, s. 897–911, 2018.

[2] S. Etigowni, D. J. Tian, G. Hernandez, S. Zonouz i K. Butler, „CPAC: Securing Critical Infrastructure with Cyber-Physical Access Control”, w *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16, New York, NY, USA: Association for Computing Machinery, 2016, 139–152, ISBN: 9781450347716. DOI: 10.1145/2991079.2991126. adr.: https://doi.org/10.1145/2991079.2991126.

[3] N. Miloslavskaya, „SOC- and SIC-Based Information Security Monitoring”, w *Recent Advances in Information Systems and Technologies*, Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis i S. Costanzo, red., Cham: Springer International Publishing, 2017, s. 364–374, ISBN: 978-3-319-56538-5.

[4] *Verizon 2019 data breach investigation report*, https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf, Accessed: 2020-07-30.

[5] M. Dowd, J. McDonald i J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006, ISBN: 0321444426.

[6] K. Price, „Vulnerability Management in Software: Before Patch Tuesday”, w *BlackHat*, t. NA, lip. 2015, NA–NA. DOI: 0.

[7] S. Planning, „The economic impacts of inadequate infrastructure for software testing”, *National Institute of Standards and Technology*, 2002.

[8] D. G. Feitelson, E. Frachtenberg i K. L. Beck, „Development and Deployment at Facebook”, *IEEE Internet Computing*, t. 17, nr. 4, s. 8–17, 2013.

[9] E. Ries, *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Currency, 2011.

[10] K. Scarfone i P. Mell, „An analysis of CVSS version 2 vulnerability scoring”, w *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2009, s. 516–525.

[11] G. Siewruk i W. Mazurczyk, „Context-Aware Software Vulnerability Classification Using Machine Learning”, *IEEE Access*, t. 9, s. 88 852–88 867, 2021.

[12] C. Chio i D. Freeman, *Machine learning and security: Protecting systems with data and algorithms*. " O'Reilly Media, Inc.", 2018.

[13] T. C. Truong, I. Zelinka, J. Plucar, M. Čandík i V. Šulc, „Artificial intelligence and cybersecurity: Past, presence, and future”, w *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, Springer, 2020, s. 351–363.

[14] A. Mathews, „What can machine learning do for information security?”, *Network Security*, t. 2019, nr. 4, s. 15–17, 2019, ISSN: 1353-4858. DOI: https://doi.org/

10.1016/S1353-4858(19)30050-9. adr.: https://www.sciencedirect.com/science/article/pii/S1353485819300509.

[15] H. Zimmermann, „OSI reference model-the ISO model of architecture for open systems interconnection", *IEEE Transactions on communications*, t. 28, nr. 4, s. 425–432, 1980.

[16] L. Allodi i F. Massacci, „Security events and vulnerability data for cybersecurity risk estimation", *Risk Analysis*, t. 37, nr. 8, s. 1606–1627, 2017.

[17] O. Ibitoye, R. Abou-Khamis, A. Matrawy i M. O. Shafiq, *The Threat of Adversarial Attacks on Machine Learning in Network Security – A Survey*, 2020. arXiv: 1911.02621 [cs.CR].

[18] T. Saranya, S. Sridevi, C. Deisy, T. D. Chung i M. Khan, „Performance Analysis of Machine Learning Algorithms in Intrusion Detection System: A Review", *Procedia Computer Science*, t. 171, s. 1251–1260, 2020, Third International Conference on Computing and Network Communications (CoCoNet'19), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.133. adr.: https://www.sciencedirect.com/science/article/pii/S1877050920311121.

[19] J. Singh i M. J. Nene, „A survey on machine learning techniques for intrusion detection systems", *International Journal of Advanced Research in Computer and Communication Engineering*, t. 2, nr. 11, s. 4349–4355, 2013.

[20] M. Almseidin, M. Alzubi, S. Kovacs i M. Alkasassbeh, „Evaluation of machine learning algorithms for intrusion detection system", w *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, IEEE, 2017, s. 000 277–000 282.

[21] P. Laskov, P. Düssel, C. Schäfer i K. Rieck, „Learning intrusion detection: supervised or unsupervised?", w *International Conference on Image Analysis and Processing*, Springer, 2005, s. 50–57.

[22] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang i K. Li, „HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning", *Computer Networks*, t. 169, s. 107 049, 2020, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2019.107049. adr.: https://www.sciencedirect.com/science/article/pii/S1389128619304086.

[23] S. U. Jan, S. Ahmed, V. Shakhov i I. Koo, „Toward a lightweight intrusion detection system for the internet of things", *IEEE Access*, t. 7, s. 42 450–42 471, 2019.

[24] M. Hasan, M. M. Islam, M. I. I. Zarif i M. Hashem, „Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches", *Internet of Things*, t. 7, s. 100 059, 2019.

[25] T Saranya, S Sridevi, C Deisy, T. D. Chung i M. A. Khan, „Performance analysis of machine learning algorithms in intrusion detection system: A review", *Procedia Computer Science*, t. 171, s. 1251–1260, 2020.

[26] L. F. Maimó, Á. L. P. Gómez, F. J. G. Clemente, M. G. Pérez i G. M. Pérez, „A self-adaptive deep learning-based system for anomaly detection in 5G networks”, *IEEE Access*, t. 6, s. 7700–7712, 2018.

[27] S. Saad, W. Briguglio i H. Elmiligi, „The Curious Case of Machine Learning In Malware Detection”, *CoRR*, t. abs/1905.07573, 2019. arXiv: 1905.07573. adr.: http://arxiv.org/abs/1905.07573.

[28] B. Chu, T. J. Holt i G. J. Ahn, „Examining the creation, distribution, and function of malware on-line”, *Department of Justice Abstract*, s. 1–183, 2010.

[29] A. O. Almashhadani, M. Kaiiali, S. Sezer i P. O'Kane, „A Multi-Classifier Network-Based Crypto Ransomware Detection System: A Case Study of Locky Ransomware”, *IEEE Access*, t. 7, s. 47 053–47 067, 2019. DOI: 10.1109/ACCESS.2019.2907485.

[30] M. K. Alzaylaee, S. Y. Yerima i S. Sezer, „DL-Droid: Deep learning based android malware detection using real devices”, *Computers & Security*, t. 89, s. 101 663, 2020, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.101663. adr.: https://www.sciencedirect.com/science/article/pii/S0167404819300161.

[31] C. Islam, M. Ali Babar i S. Nepal, „A Multi-Vocal Review of Security Orchestration”, *ACM Computing Surveys*, t. 52, s. 1–45, kw. 2019. DOI: 10.1145/3305268.

[32] *Komand*, https://komand.com, Accessed: 2020-07-30.

[33] *Intel Security, Automating the Threat Defence Lifecycle*, https://www.mcafee.com/au/solutions/orchestration.aspx, Accessed: 2020-07-30.

[34] T. Koyama, B. Hu, Y. Nagafuchi, E. Shioji i K. Takahashi, „Security orchestration with a global threat intelligence platform”, t. 13, grud. 2015.

[35] *Faraday Sec*, https://www.faradaysec.com, Accessed: 2020-07-30.

[36] *OWASP Defect Dojo*, https://owasp.org/www-project-defectdojo/, Accessed: 2020-07-30.

[37] J. A. Harer, L. Y. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, L. H. Hamilton, G. I. Centeno, J. R. Key, P. M. Ellingwood, E. Antelman, A. Mackay, M. W. McConley, J. M. Opper i P. Chin, „Automated software vulnerability detection with machine learning”, 2018.

[38] Z. L. D. Zou, S. Xu, H. Jin, S. member, Y. Zhu i Z. Chen, „SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities”, 2018.

[39] A. Hovsepyan, R. Scandariato, W. Joosen i J. Walden, „Software vulnerability prediction using text analysis techniques.”, *Proc. 4th Int. Workshop Security Measurements and Metrics, MetriSec*, 2012.

[40] R. L. Russell, L. Kim, L. H. Hamilton, T. Lazovich, J. A. Harer, O. Ozdemir, P. M. Ellingwood i M. W. McConley, „Automated Vulnerability Detection in Source Code Using Deep Representation Learning.”, *2018 17th IEEE International Conference on Machine Learning and Applications.*, 2018.

[41] M. D. M. Zulkernine i F. Jaafar, „An Automatic Software Vulnerability Classification Framework.”, *2017 International Conference on Software Security and Assurance.*, 2017.

[42] B. S. H. Li, M. Li, Q. Zhang i C. Tang, „Automatic Classification for Vulnerability Based on Machine Learning.", *International Conference on Information and Automation Yinchuan*, 2013.

[43] Z. Han, X. Li, Z. Xing, H. Liu i Z. Feng, „Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description", w *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, s. 125–136. DOI: 10.1109/ICSME.2017.52.

[44] O. H. Alhazmi i Y. K. Malaiya, „Prediction capabilities of vulnerability discovery models", w *RAMS'06. Annual Reliability and Maintainability Symposium, 2006.*, IEEE, 2006, s. 86–91.

[45] S. Zhang, D. Caragea i X. Ou, „An empirical study on using the national vulnerability database to predict software vulnerabilities", w *International conference on database and expert systems applications*, Springer, 2011, s. 217–231.

[46] P. Frijns, R. Bierwolf i T. Zijderhand, „Reframing Security in Contemporary Software Development Life Cycle.", *IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD)*, 2018.

[47] A. Daniels i D. Yeates, *Basic training in systems analysis*. Pitman [for] the National Computing Centre, 1971, ISBN: 0273316567.

[48] S. Radack, „The system development life cycle (sdlc)", National Institute of Standards i Technology, spraw. tech., 2009.

[49] M. Khari, Vaishali i P. Kumar, „Embedding security in Software Development Life Cycle (SDLC)", w *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, s. 2182–2186.

[50] W. E. Lewis, *Software testing and continuous quality improvement*. CRC press, 2017.

[51] J. de Vicente Mohino, J. Bermejo Higuera, J. R. Bermejo Higuera i J. A. Sicilia Montalvo, „The application of a new secure software development life cycle (S-SDLC) with agile methodologies", *Electronics*, t. 8, nr. 11, s. 1218, 2019.

[52] H. Assal i S. Chiasson, „Security in the Software Development Lifecycle", w *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, Baltimore, MD: USENIX Association, sierp. 2018, s. 281–296, ISBN: 978-1-939133-10-6. adr.: https://www.usenix.org/conference/soups2018/presentation/assal.

[53] P. C. Jorgensen, *Software testing: a craftsman's approach*. CRC press, 2018.

[54] C. Ward i L. Legorreta, „Beyond waterfall and agile methods: towards a new contingency model for IT project management", *Available at SSRN 1400254*, 2009.

[55] Y. B. Leau, W. K. Loo, W. Y. Tham i S. F. Tan, „Software development life cycle AGILE vs traditional approaches", w *International Conference on Information and Network Technology*, t. 37, 2012, s. 162–167.

[56] M. Kramer, „Best practices in systems development lifecycle: An analyses based on the waterfall model", *Review of Business & Finance Studies*, t. 9, nr. 1, s. 77–84, 2018.

[57] K. B. Hass, „The blending of traditional and agile project management", *PM world today*, t. 9, nr. 5, s. 1–8, 2007.

[58] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries i in., „Manifesto for agile software development", 2001.

[59] S Balaji i M. S. Murugaiyan, „Waterfall vs. V-Model vs. Agile: A comparative study on SDLC", *International Journal of Information Technology and Business Management*, t. 2, nr. 1, s. 26–30, 2012.

[60] J. Luszcz, „Apache Struts 2: how technical and development gaps caused the Equifax Breach", *Network Security*, t. 2018, nr. 1, s. 5–8, 2018.

[61] J. Li, „Vulnerabilities Mapping based on OWASP-SANS: A Survey for Static Application Security Testing (SAST)", 2020.

[62] T. Rangnau, R. v. Buijtenen, F. Fransen i F. Turkmen, „Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines", w *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, 2020, s. 145–154. DOI: 10.1109/EDOC49727.2020.00026.

[63] Y. Pan, „Interactive Application Security Testing", w *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, IEEE, 2019, s. 558–561.

[64] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[65] A. L. Samuel, „Some studies in machine learning using the game of checkers", *IBM Journal of research and development*, t. 3, nr. 3, s. 210–229, 1959.

[66] T. O. Ayodele, „Types of machine learning algorithms", *New advances in machine learning*, t. 3, s. 19–48, 2010.

[67] F. Osisanwo, J. Akinsola, O Awodele, J. Hinmikaiye, O Olakanmi i J Akinjobi, „Supervised machine learning algorithms: classification and comparison", *International Journal of Computer Trends and Technology (IJCTT)*, t. 48, nr. 3, s. 128–138, 2017.

[68] L. Breiman, J. Friedman, C. J. Stone i R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[69] J. Friedman, T. Hastie, R. Tibshirani i in., *The elements of statistical learning, Wyd 2*, 10. Springer series in statistics New York, 2008, t. 1.

[70] L. Breiman, „Random forests", *Machine learning*, t. 45, nr. 1, s. 5–32, 2001.

[71] E. Riloff, R. Jones i in., „Learning dictionaries for information extraction by multi-level bootstrapping", w *AAAI/IAAI*, 1999, s. 474–479.

[72] W. S. McCulloch i W. Pitts, „A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, t. 5, nr. 4, s. 115–133, 1943.

[73] B. Widrow i M. E. Hoff, „Adaptive switching circuits", Stanford Univ Ca Stanford Electronics Labs, spraw. tech., 1960.

[74] L. R. Medsker i L. Jain, „Recurrent neural networks", *Design and Applications*, t. 5, 2001.

[75] S. Albawi, T. A. Mohammed i S. Al-Zawi, „Understanding of a convolutional neural network", w *2017 International Conference on Engineering and Technology (ICET)*, Ieee, 2017, s. 1–6.

[76] M. Mathieu, C. Couprie i Y. LeCun, „Deep multi-scale video prediction beyond mean square error”, *arXiv preprint arXiv:1511.05440*, 2015.

[77] P.-T. De Boer, D. P. Kroese, S. Mannor i R. Y. Rubinstein, „A tutorial on the cross-entropy method”, *Annals of operations research*, t. 134, nr. 1, s. 19–67, 2005.

[78] W. Yin, K. Kann, M. Yu i H. Schütze, *Comparative Study of CNN and RNN for Natural Language Processing*, 2017. arXiv: 1702.01923 [cs.CL].

[79] A. Conneau, H. Schwenk, L. Barrault i Y. Lecun, „Very deep convolutional networks for natural language processing”, *arXiv preprint arXiv:1606.01781*, t. 2, 2016.

[80] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, V. Vapnik i in., „Support vector regression machines”, *Advances in neural information processing systems*, t. 9, s. 155–161, 1997.

[81] B. E. Boser, I. M. Guyon i V. N. Vapnik, „A training algorithm for optimal margin classifiers”, w *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, s. 144–152.

[82] V. Vapnik, „The support vector method of function estimation”, w *Nonlinear modeling*, Springer, 1998, s. 55–85.

[83] F. H. Clarke, „A new approach to Lagrange multipliers”, *Mathematics of Operations Research*, t. 1, nr. 2, s. 165–174, 1976.

[84] G. F. Smits i E. M. Jordaan, „Improved SVM regression using mixtures of kernels”, w *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, IEEE, t. 3, 2002, s. 2785–2790.

[85] R. M. French, „The Turing Test: the first 50 years”, *Trends in cognitive sciences*, t. 4, nr. 3, s. 115–122, 2000.

[86] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz i in., „HuggingFace's Transformers: State-of-the-art natural language processing”, *arXiv preprint arXiv:1910.03771*, 2019.

[87] L. Deng i Y. Liu, *Deep learning in natural language processing*. Springer, 2018.

[88] U. Kamath, J. Liu i J. Whitaker, *Deep learning for NLP and speech recognition*. Springer, 2019, t. 84.

[89] M. Mnasri, „Recent advances in conversational NLP: Towards the standardization of Chatbot building”, *arXiv preprint arXiv:1903.09025*, 2019.

[90] E. Cambria, S. Poria, A. Gelbukh i M. Thelwall, „Sentiment analysis is a big suitcase”, *IEEE Intelligent Systems*, t. 32, nr. 6, s. 74–80, 2017.

[91] B. McCann, J. Bradbury, C. Xiong i R. Socher, „Learned in translation: Contextualized word vectors”, *arXiv preprint arXiv:1708.00107*, 2017.

[92] Y. A. Solangi, Z. A. Solangi, S. Aarain, A. Abro, G. A. Mallah i A. Shah, „Review on Natural Language Processing (NLP) and Its Toolkits for Opinion Mining and Sentiment Analysis”, w *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2018, s. 1–4. DOI: 10.1109/ICETAS.2018.8629198.

[93] S. Chiaretta, *Front-end Development with ASP. NET Core, Angular, and Bootstrap*. John Wiley & Sons, 2018.

[94] A. Balalaie, A. Heydarnoori i P. Jamshidi, „Microservices architecture enables devops: Migration to a cloud-native architecture", *Ieee Software*, t. 33, nr. 3, s. 42–52, 2016.

[95] N. Jatana, S. Puri, M. Ahuja, I. Kathuria i D. Gosain, „A survey and comparison of relational and non-relational database", *International Journal of Engineering Research & Technology*, t. 1, nr. 6, s. 1–5, 2012.

[96] C. Wijayarathna i N. A. Arachchilage, „Why Johnny Can't Store Passwords Securely? A Usability Evaluation of Bouncycastle Password Hashing", w *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, 2018, s. 205–210.

[97] J. Topper, „Compliance is not security", *Computer Fraud & Security*, t. 2018, nr. 3, s. 5–8, 2018.

[98] B. De, „Api Management", w *API Management*, Springer, 2017, s. 15–28.

[99] V. Surwase, „REST API modeling languages-a developer's perspective", *Int. J. Sci. Technol. Eng*, t. 2, nr. 10, s. 634–637, 2016.

[100] *ThreadFix*, `https://threadfix.it`, Accessed: 2021-02-08.

[101] *CodeDX*, `https://codedx.com`, Accessed: 2021-02-08.

[102] M. J. Wallent, R. Dujari, A. Ramakrishna, L. M. Kohnfelder i L. Geer, *Network security zones*, US Patent 6,366,912, kw. 2002.

[103] C. Anderson, „Docker [software engineering]", *Ieee Software*, t. 32, nr. 3, s. 102–c3, 2015.

[104] I. Chalvatzis, D. A. Karras i R. C. Papademetriou, „Evaluation of security vulnerability scanners for small and medium enterprises business networks resilience towards risk assessment", w *2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, IEEE, 2019, s. 52–58.

[105] C. Joshi i U. K. Singh, „Security testing and assessment of vulnerability scanners in quest of current information security landscape", *International Journal of Computer Applications*, t. 145, nr. 2, s. 1–7, 2016.

[106] A. Z. Baset i T. Denning, „Ide plugins for detecting input-validation vulnerabilities", w *2017 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2017, s. 143–146.

[107] J. C. Chang, S. Amershi i E. Kamar, „Revolt: Collaborative Crowdsourcing for Labeling Machine Learning Datasets", w *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2017, 2334–2346, ISBN: 9781450346559. adr.: `https://doi.org/10.1145/3025453.3026044`.

[108] S. Norberg, „Secure Application Life Cycle Management", w *Advanced ASP. NET Core 3 Security*, Springer, 2020, s. 373–396.

[109] M. Ettredge, J. Heintz, C. Li i S. Scholz, „Auditor realignments accompanying implementation of SOX 404 ICFR reporting requirements", *Accounting Horizons*, t. 25, nr. 1, s. 17–39, 2011.

[110] C. Gikas, „A general comparison of fisma, hipaa, iso 27000 and pci-dss standards", *Information Security Journal: A Global Perspective*, t. 19, nr. 3, s. 132–141, 2010.

[111] W. Etaiwi i G. Naymat, „The impact of applying different preprocessing steps on review spam detection", *Procedia computer science*, t. 113, s. 273–279, 2017.

[112] S. L. Özesmi, C. O. Tan i U. Özesmi, „Methodological issues in building, training, and testing artificial neural networks in ecological applications", *Ecological Modelling*, t. 195, nr. 1-2, s. 83–93, 2006.

[113] J. Subramanian i R. Simon, „Overfitting in prediction models–is it a problem only in high dimensions?", *Contemporary clinical trials*, t. 36, nr. 2, s. 636–641, 2013.

[114] J. Brownlee, *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras*. Machine Learning Mastery, 2016.

[115] P. Lison, „An introduction to machine learning", *Language Technology Group (LTG)*, t. 1, nr. 35, 2015.

[116] C. Thornton, F. Hutter, H. H. Hoos i K. Leyton-Brown, „Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms", w *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13, New York, NY, USA: Association for Computing Machinery, 2013, 847–855, ISBN: 9781450321747. DOI: 10.1145/2487575.2487629. adr.: https://doi.org/10.1145/2487575.2487629.

[117] K. Duan, S. S. Keerthi i A. N. Poo, „Evaluation of simple performance measures for tuning SVM hyperparameters", *Neurocomputing*, t. 51, s. 41–59, 2003.

[118] L. N. Smith, „A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay", *arXiv preprint arXiv:1803.09820*, 2018.

[119] S. Sharma i S. Sharma, „Activation functions in neural networks", *Towards Data Science*, t. 6, nr. 12, s. 310–316, 2017.

[120] P. Probst, M. N. Wright i A.-L. Boulesteix, „Hyperparameters and tuning strategies for random forest", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, t. 9, nr. 3, e1301, 2019.

[121] K. Stąpor, „Evaluating and comparing classifiers: Review, some recommendations and limitations", w *International Conference on Computer Recognition Systems*, Springer, 2017, s. 12–21.

[122] N. Seliya, T. M. Khoshgoftaar i J. Van Hulse, „A Study on the Relationships of Classifier Performance Metrics", w *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, 2009, s. 59–66. DOI: 10.1109/ICTAI.2009.25.

[123] I. Myrtveit, E. Stensrud i M. Shepperd, „Reliability and validity in comparative studies of software prediction models", *IEEE Transactions on Software Engineering*, t. 31, nr. 5, s. 380–391, 2005.

[124] H. Caselles-Dupré, F. Lesaint i J. Royo-Letelier, „Word2vec applied to recommendation: Hyperparameters matter”, w *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018, s. 352–356.

[125] D. Luna, L. Allodi i M. Cremonini, „Productivity and patterns of activity in bug bounty programs: Analysis of HackerOne and Google vulnerability research”, w *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, s. 1–10.

Spis załączników

1. Instrukcja pozwalająca na odtworzenie badania 107
2. Nagroda szefa funkcji Sieć i Technologie 2019 110
3. Privacy & Security Awards 2021 111

Załącznik 1. Instrukcja pozwalająca na odtworzenie badania

Wszystkie komponenty systemy Mixeway są udostępnione na podstawie licencji MIT za pośrednictwem serwisu GitHub. Kod źródłowy aplikacji, dzięki której możliwe było uzyskanie zaprezentowanych wyników znajduję się w repozytorium aplikacji *Mixeway-VulnAuditor* pod adresem `https://github.com/mixeway/mixewayvulnauditor`.

Omawiana aplikacja może być uruchomiona na 3 sposoby:

- Jako zestaw testowy - weryfikujący skuteczność algorytmów takich jak Sieci Neuronowe, Wektor Maszyny Nośnych czy Las Losowy na podanym zestawie danych testowych
- Jako aplikacja klasyfikująca - aplikacyjny interfejs programistyczny, pozwalający na oznaczenie podanej podatności jako "CRV" lub "DRNV"

Aby uruchomić aplikacje jako zestaw testowy, należy wcześniej przygotować zestawy danych, które zostaną wykorzystane najpierw do wytrenowania modelu, a następnie do jego przetestowania. Zestaw danych musi znajdować się w lokalizacji **data** w formacie **CSV** (możliwe jest umieszczeine wielu plików CSV we wskazanej lokalizacji). Pliki z podatnościami muszą zawierać nagłówek oraz kolumny: *app_name, app_context, vuln_name, vuln_desc, severity, grade* gdzie:

- app_name: nazwa aplikacji, w której została wykryta podatność. W przypadku skanu typu SAST jest to nazwa repozytorium kodu, w przypadku skany typu DAST jest to adres URL
- app_context: kontekst, w jakim działa aplikacja. W badanym eksperymencie pole zawiera format klucz wartość, np.: *type webappvuln location localdc usedby customer domain internet* czyli: podatność znaleziona w aplikacji internetowej, działającej w infrastrukturze firmy dostępnej z sieci Internet dla klientów firmy.
- vuln_name: Nazwa podatności zgłoszona przez skaner podatności np.: Cross-site Scripting
- vuln_description: Opis podatności zgłoszony przez skaner podatności oraz jeśli jest to dostępne zalecania dotyczące naprawienia podatności (jeśli takie dane są udostępniane przez oprogramowanie testujące)
- severity: Krytyczność zgłoszona przez skaner podatności
- grade: Ocena czy podatność należy do klasy "CRV" (1) czy "DRNV" (0)

Aby uruchomić zestaw testowy, należy wykonać operacje, która uruchomi aplikację:

```
py ./src/main/MixewayVulnAuditor.py
```

Przykładowy wynik, wyświetlany na konsoli po uruchomieniu aplikacji został przedstawiony na Rysunku Załącznik 1.1.

```
Loading file:  C:\Users\gsiew\IdeaProjects\MixewayVulnAuditor\data\audit_code.csv
Loading file:  C:\Users\gsiew\IdeaProjects\MixewayVulnAuditor\data\audit_infra.csv
Loading file:  C:\Users\gsiew\IdeaProjects\MixewayVulnAuditor\data\audit_os.csv
Loading file:  C:\Users\gsiew\IdeaProjects\MixewayVulnAuditor\data\audit_webapp.csv
Prepared dictionary for tokenizer
Train set size:  37737
Train set Labels:  {0: 24254, 1: 13483}
Test set size:  16173
Test set Labels:  {0: 10384, 1: 5789}
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 863, 32)           607264
_________________________________________________________________
flatten (Flatten)            (None, 27616)             0
_________________________________________________________________
dense (Dense)                (None, 32)                883744
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 16)                528
_________________________________________________________________
dense_3 (Dense)              (None, 16)                272
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,492,881
Trainable params: 1,492,881
Non-trainable params: 0
_________________________________________________________________
Layers:  7
Epoch 1/50
1180/1180 [==============================] - 17s 14ms/step - loss: 0.2768 - accuracy: 0.9699 - f1:
Epoch 2/50
```

Rys. Załącznik 1.1: Przykładowy wynik uruchomienia aplikacji testującej

Aby uruchomić aplikację w trybie serwera, wymagane jest wcześniejsze opracowanie modelu, który będzie wykorzystany do klasyfikacji przykładów, które zostaną przesłane do aplikacji przez API. Aby uruchomić rozwiązanie w tym trybie, należy wykonać operacje:

```
py ./src/main/vuln_auditor_server.py
```

Udostępnione API zawiera jedną metodę, która przyjmuje obiekt opisujący podatność:

```
POST http://localhost:8445/vuln/perdict
[
    {
    "id": 1,
     "appName": "Apollo",
     "appContext": "type opensource customer internal location remotedc
domain internal",
     "vulnName": "CVE-2016-1000339",
     "vulnDescription": "In the Bouncy Castle JCE Provider version 1.55
and earlier the primary engine class used for AES was AESFastEngine. Due
to the highly table driven approach used in the algorithm it turns out that
if the data channel on the CPU can be monitored the lookup table accesses
are sufficient to leak information on the AES key being used. There was
also a leak in AESEngine although it was substantially less. XXEOS has been
modified to remove any signs of leakage (testing carried out on Intel X86-64)
and is now the primary AES class for the BC JCE provider from 1.56. Use
of AESFastEngine is now only recommended where otherwise deemed appropriate.",
     "severity": "Medium"
    }
]
```

odpowiedź to:

```
[
    {
        "id": 1,
        "audit": 0
    }
]
```

Załącznik 2. Nagroda szefa funkcji Sieć i Technologie 2019

orange

Dyplom Laureata Nagrody Szefa Funkcji Sieć i Technologie

Grzegorz Siewruk

Grzegorzu,

Dziękuję za opracowanie koncepcji, a następnie wdrożenie kompletnego rozwiązania automatyzującego proces wykrywania podatności na zagrożenia w komponentach wykorzystywanych w procesach programistycznych (kod, infrastruktura i zależności OpenSource) - Mixeway. Jest to oprogramowanie, którego zadaniem jest koordynowanie skanowania w poszukiwaniu podatności wykorzystywanych w procesach OPL DevSecOps.

Liczę na Twoje dalsze, pełne zaangażowanie w realizację naszych celów biznesowych.
Serdecznie gratuluję!

Piotr Jaworski

Członek Zarządu ds. Sieci i Technologii

Warszawa, 13.05.2020r.

Załącznik 3. Privacy & Security Awards 2021