Uniwersytet Technologiczno-Przyrodniczy
im. Jana i Jędrzeja Śniadeckich w Bydgoszczy

**Wydział Telekomunikacji
Informatyki i Elektrotechniki**

ROZPRAWA DOKTORSKA

mgr inż. Marek Pawlicki

Zastosowanie Metod Uczenia Maszynowego do Wykrywania Ataków Sieciowych

Promotor
dr hab. inż. Michał Choraś, prof. uczelni

Promotor pomocniczy
dr hab. inż. Rafał Kozik, prof. uczelni

Bydgoszcz 2020

Spis treści

0.1. Tablica Skrótów

Tablica 1: Tablica Skrótów.

j. angielski	skrót	j. polski
Accuracy	ACC	skuteczność, trafność, celność
Adaptive Moment Estimation	ADAM	adaptacyjna estymacja momentu
Adaptive Boosting	ADABoost	
Artificial Intelligence	AI	sztuczna inteligencja
Artificial Immune System	AIS	sztuczny system immunologiczny
Artificial Neural Networks	ANN	sztuczne sieci neuronowe
Bidirectional Recurrent Neural Network	BRNN	dwukierunkowe rekurencyjne sieci neuronowe
Canadian Institute for Cybersecurity Intrusion Detection Systems	CICIDS	systemy wykrywania ataków sieciowych Kanadyjskiego Instytutu Cyberbezpieczeństwa
Convolutional Neural Network	CNN	konwolucyjna sieć neuronowa, splotowa sieć neuronowa
Convolutional Recurrent Neural Network	CRNN	konwolucyjna rekurencyjna sieć neuronowa
Critical Infrastructure	CI	infrastruktura krytyczna
Cross-Site Scripting	XSS	skrypt międzywitrynowy
Deep Confidence Neural Network	DCNN	
Distributed Denial of Service	DDoS	rozproszona odmowa usługi
Deep Learning	DL	głębokie uczenie
False Positive / False Positive Rate	FP / FPR	(współczynnik) wynik fałszywie dodatni
False Negative	FN	wynik fałszywie ujemny
Gated Recurrent Unit	GRU	
Generative Adversarial Network	GAN	generatywne sieci współzawodniczące
Hard hyperbolic tangent	Hard Tanh	twardy tangens hiperboliczny
Hard Sigmoid	Hard Sig	twarda funkcja sigmoidalna
Host-Based Intrusion System	HIDS	wykrywanie ataków sieciowych oparte o hosta

Ciąg dalszy tab. 1		
Hierarchical Spatial-Temporal Features-Based Intrusion Detection System	HAST-IDS	hierarchiczny, czasoprzestrzenny system wykrywania ataków sieciowych
Internet of Things	IoT	internet rzeczy
Intrusion Detection System	IDS	systemy wykrywania ataków sieciowych
Knowledge Discovery in Databases 1999	KDD'99	odkrywanie wiedzy w zbiorach danych
K-Nearest Neighbour	K-NN	k najbliższych sąsiadów
K-Means		algorytm centroidów, k-średnich
Linear Discriminant Analysis	LDA	liniowa analiza dyskryminacyjna
Linear Regression	LR	regresja liniowa
Naive Bayes Classifier	NB	naiwny klasyfikator bayesowski
Natural Language Processing	NLP	przetwarzanie języka naturalnego
Network-Based Intrusion System	NIDS	wykrywanie ataków sieciowych w oparciu o analizę ruchu sieciowego
Machine Learning	ML	uczenie maszynowe
Multi-Layer Perceptron	MLP	perceptron wielowarstwowy
Particle Swarm Optimisation	PSO	optymalizacja rojem cząstek
Principal Component Analysis	PCA	analiza głównych komponentów
Random Forest	RF	losowy las
Rectified Linear Unit	ReLU	obcięta funkcja liniowa
Recurrent Neural Network	RNN	rekurencyjna sieć neuronowa
Restricted Boltzman Machine	RBM	ograniczona maszyna Boltzmana
Root Mean Square Propagation	rmsprop	propagacja pierwiastka średnich kwadratów
Security Information and Event Management	SIEM	
Sigmoid	sig	funkcja sigmoidalna

Ciąg dalszy tab. 1		
Synthetic Minority Oversampling Technique	SMOTE	technika syntetycznego oversamplingu mniejszości
Security operations center	SOC	
Software Defined Network	SDN	
Stochastic Gradient Descent	SGD	spadek stochastyczny gradientu, zrównoleglona optymalizacja stochastyczna
Support Vector Machine	SVM	maszyna wektorów nośnych
True Negative	TN	wynik prawdziwie ujemny
True Positive	TP	wynik prawdziwie dodatni
Koniec Tablicy Skrótów		

1. Wstęp

1.1. Motywacja

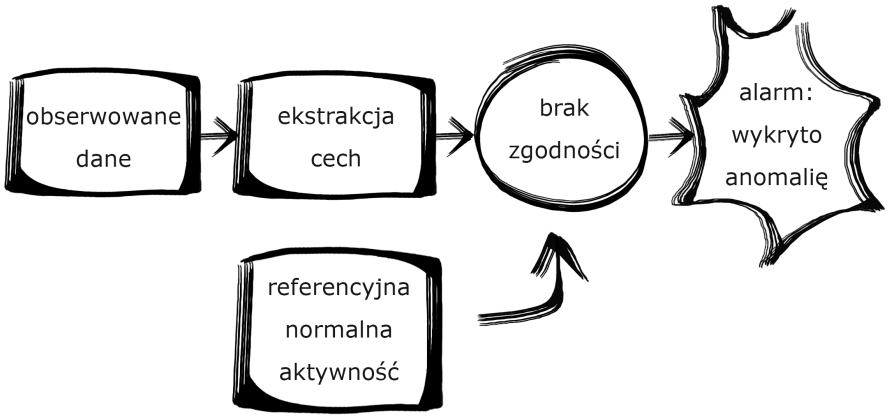
Domena cyberbezpieczeństwa jest dojrzałą dziedziną z szeregiem komercyjnych produktów i wachlarzem znanych rozwiązań. Pomimo to każdego dnia tak firmy jak i obywatele otoczeni są całą gamą takich cyberzagrożeń jak np. złośliwe oprogramowanie, konie trojańskie, spyware, ataki typu SQL injection, cross-site scripting, ransomware [71] i wiele, wiele innych. W pewnym sensie wszystkie te wrogo nastawione elementy cyberprzestrzeni stały się po prostu częścią codzienności. Całkiem niedawno, na początku 2018 roku, skierowane przeciwko Androidowi złośliwe oprogramowanie bankowe splądrowało portfele nic niepodejrzewających użytkowników aplikacji finansowej. Nowa odmiana BankBota została zmodyfikowana do takiego stopnia, że zdołała ominąć zabezpieczenie antywirusowe Sklepu Google Play, mimo że BankBot jest doskonale znanym złośliwym oprogramowaniem. Trojan działał pod przykrywką na pozór nieszkodliwej aplikacji, lecz wkrótce po zainstalowaniu na urządzeniu z Androidem zaczynał wykradać dane dostępu do banku [8].

Bardzo znane zastosowanie skryptu międzywitrynowego, tzw. XSS (*ang. cross-site scripting*) miało miejsce kiedy eBay wykazał się brakiem zabezpieczeń przed takim rodzajem ataków [49]. Kod JavaScript dodawano do stron z ofertami kosztownych przedmiotów. Użytkownik musiał tylko kliknąć złośliwą, ale wyglądającą niewinnie ofertę, aby pozwolić skryptowi na przejęcie kontroli nad jego przeglądarką, która przekierowywała go do strony wyglądającej identycznie jak eBay, lecz wykradającej dane karty kredytowej.

Na początku roku 2018 miało miejsce naruszenie bezpieczeństwa popularnej aplikacji fitness, *MyFitnessPal*. Zdarzenie to dotknęło ponad 150 milionów jej użytkowników. Media zajmujące się incydentem próbowały przedstawić zdarzenie jako "po prostu kolejny dzień w Internecie" [62]. Przy obecnie występującym ryzyku ze strony tak nowych jak i znanych zagrożeń cyberbezpieczeństwa, kuszące jest by po prostu przytaknąć tego typu twierdzeniom.

Ogromna liczba i zakres naruszeń cyberbezpieczeństwa zaowocowały powstawaniem szeregu rozmaitych metod ich wykrywania. W dziedzinie badań i rozwoju zaobserwowano dwa trendy - podejście bazujące na sygnaturach ataków i podejście badające anomalie. IDS (systemy wykrywania ataków sieciowych, ang. *Intrusion Detection Systems*) działają w oparciu o repozytorium rozpoznanych ataków, natomiast metody oparte o anomalie tworzą model "normalnego" ruchu sieciowego i wszczynają alarm w wypadku zauważenia każdego odstępstwa od tego modelu [45].

Środowisko hakerów stosuje rozmaite techniki obfuskacji aby ominąć detektory oparte na sygnaturach. W świetle najnowszych analiz, znane złośliwe oprogramowanie może zostać skutecznie ukryte [16].



Rysunek 1. Wykrywanie ataków sieciowych w oparciu o detekcję anomalii

Bezpieczeństwo dzisiejszych systemów cyfrowych jest zagadnieniem o krytycznej wadze w szerokiej gamie gałęzi przemysłu, zaczynając od bankowego, przez komunikację, finanse, działania organów ścigania, przemysł medyczny a nawet podbój kosmosu. Wykrywanie ataków sieciowych jest jednym z zadań monitorowania i ewaluacji ruchu sieciowego w poszukiwaniu jakichkolwiek oznak naruszeń bezpieczeństwa. Przy popularyzacji użycia komputerów, cyberbezpieczeństwo stale zyskuje na wadze [69].

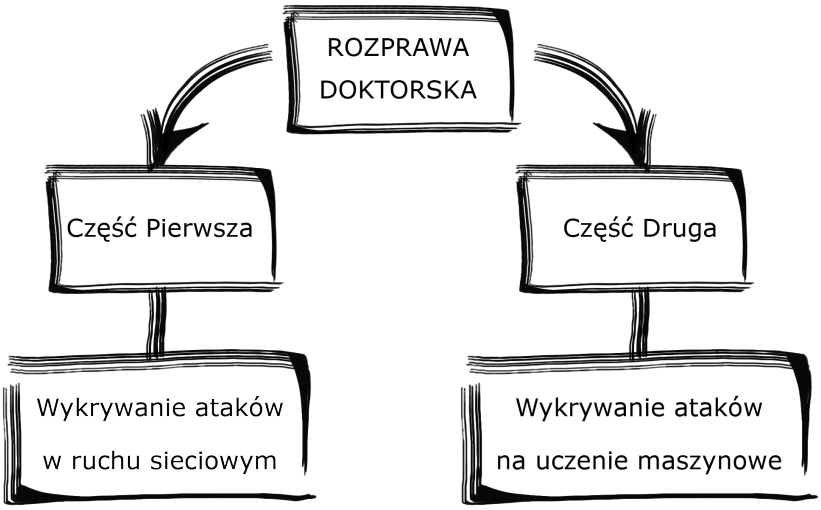
W 70 miliardach wydarzeń związanych z cyberbezpieczeństwem raportowanych przez IBM security task force każdego dnia [1], ponad połowa opiera się o dziury w systemach operacyjnych, a prawie jedna trzecia to "*spearfishing*" - atak skierowany przeciwko konkretnej osobie. Wszystkie te działania mieszczą się w kategorii ataków sieciowych.

1.2. Teza, cel i zakres pracy

W ramach rozprawy zaprezentowano rozwiązania dotyczące zastosowania procedury opartej o metody uczenia maszynowego do wykrywania ataków sieciowych. W ramach pracy poruszono dwie różne grupy ataków sieciowych - ataki mające na celu zaburzenie poufności, integralności lub dostępności systemu, oraz zupełnie nowy rodzaj zagrożeń, biorących na cel sam system wykrywania ataków sieciowych, który jest oparty o metody uczenia maszynowego. Z przeprowadzonej analizy literatury w połączeniu z wykonanymi badaniami sformułowano następujące tezę i cel rozprawy:

Tezą pracy jest: Możliwe jest opracowanie złożonych algorytmów uczenia maszynowego dla skutecznej detekcji ataków na podstawie przepływów sieciowych oraz opracowanie technik poprawy ich wiarydgodnosi i bezpieczeństwa (wykrywania ataków typu adversarial learning).

Celem rozprawy jest propozycja zmodyfikowanej metody przeznaczonej do wykrywania ataków sieciowych opartej o uczenie maszynowe oraz propozycja metody wykrywania ataków na uczenie maszynowe. Strukturę pracy zilustrowano na rys. 2



Rysunek 2. Dwie główne części niniejszej pracy

Niniejsza rozprawa powstała przy aktywnym udziale autora w realizacji trwających, międzynarodowych projektów badawczych w ramach Programu Horyzont 2020, między innymi: H2020 Infrastress i H2020 Sparta.

1.3. Wybrane publikacje w temacie rozprawy

W ramach realizacji założeń projektów i pracy naukowej autora opublikowano w języku angielskim:

- 12 recenzowanych publikacji z zakresu wykorzystania metod uczenia maszynowego w wykrywaniu ataków sieciowych i w kilku innych aplikacjach,
- Dwie w czasopismach o Impact Factor 2,591 i 1,376, pozostałe na międzynarodowych konferencjach naukowych, w tym jedna na konferencji CORE A,
- Dwie na konferencjach CORE B.

W związku z tymi publikacjami autor prezentował wyniki pracy naukowej na 9 konferencjach międzynarodowych.

1. Rafał Kozik, Marek Pawlicki, Michał Choraś, Witold Pedrycz:
   Practical Employment of Granular Computing to Complex Application Layer Cyberattack Detection, in: Complexity 2019: 5826737:1-5826737:9 (2019)
2. Marek Pawlicki, Rafał Kozik, Michał Choraś:
   Artificial Neural Network Hyperparameter Optimisation for Network Intrusion Detection, in: D.-S. Huang et al. (Eds.) Intelligent Computing Theories and Application, ICIC 2019, Lecture Notes in Computer Science, Nanchang, China, LNCS 11643, 749-760, Springer, (2019)
3. Michał Choraś, Marek Pawlicki, Rafał Kozik:
   The Feasibility of Deep Learning Use for Adversarial Model Extraction in the Cybersecurity Domain., in: Yin H., Camacho D., Tino P., Tallón-Ballesteros A., Menezes R., Allmendinger R. (eds) Intelligent Data Engineering and Automated Learning – IDEAL 2019. Lecture Notes in Computer Science, vol. 11872, 353-360, Springer (2019)
4. Marek Pawlicki, Adam Marchewka, Michał Choraś, Rafał Kozik:
   Gated Recurrent Units for Intrusion Detection, in: Image Processing and Communications. Techniques, Algorithms and Applications, Advances in Intelligent Systems and Computing, vol. 1062, 141-147, Springer (2019)
5. Rafał Kozik, Marek Pawlicki, Michał Choraś:
   Cost-Sensitive Distributed Machine Learning for NetFlow-Based Botnet Activity Detection, in Security and Communication Networks 2018: 8753870:1-8753870:8 (2018)

6. Marek Pawlicki, Michał Choraś, Rafał Kozik:
   Recent Granular Computing Implementations and its Feasibility in Cybersecurity Domain, in: International Conference on Availability, Reliability and Security : 61:1-61:6, ACM (2018)
7. Rafał Kozik, Marek Pawlicki, Michał Choraś:
   Sparse Autoencoders for Unsupervised Netflow Data Classification. in: Image Processing and Communications Challenges 10, Advances in Intelligent Systems and Computing, vol. 892, 192-199, Springer (2018)

1.4. Struktura pracy

Niniejsza praca podzielona jest na dwie części - jak to zilustrowano na rys. 2. Część pierwsza rozpoczyna się z Rozdziałem 2, który jest wprowadzeniem do zagadnienia wykrywania ataków w ruchu sieciowym. W Rozdziale 3 znaleźć można propozycje modyfikacji algorytmów uczenia maszynowego wraz z propozycjami własnych metod. W Rozdziale 4 znajdują się wyniki badań części pierwszej. Część druga zaczyna się Rozdziałem 5 i wprowadza zagadnienie detekcji ataków z grupy adversarial. W Rozdziale 6 umieszczona została propozycja własnego detektora ataków typu *evasion*. W Rozdziale 7 znajdują się wyniki badań z użyciem zaproponowanego rozwiązania. Rozdział 8 zawiera wnioski końcowe.

2. Część pierwsza: Wykrywanie ataków w ruchu sieciowym

Popularność implementacji rozwiązań sieciowych w handlu, komunikacji i wielu innych zastosowaniach znacząco wzmogła zapotrzebowanie na rzetelne, niezawodne rozwiązania z zakresu cyberbezpieczeństwa. Zaniedbane kwestie cyberbezpieczeństwa dotyczące jednostek lub organizacji mogą doprowadzić do niebezpiecznych sytuacji, takich jak utrata krytycznych danych, sprzyjanie atakom (również na infrastruktury krytyczne) albo przyczynianie się do ataków typu DDoS (ang. *Distributed Denial of Service* - rozproszona odmowa usługi) [17, 121].

Bezpieczeństwo każdego systemu pozostaje w bezpośredniej proporcji do decyzji dotyczących wymogów bezpieczeństwa i ich względnej priorytetyzacji, podejmowanych zarówno przez organizacje jak i jednostki. Liczba cyberataków wzrasta od kilku lat i przyczynił się do tego cały szereg czynników. Aby mieć to zjawisko pod kontrolą, wprowadzono wiele zmian w zakresie badań i polityki, jednakże jak dotąd nie wyznaczono jeszcze jasnej drogi ku stabilnym i bezpiecznym systemom. Aby zrozumieć czym umotywowane są ataki cybernetyczne, może się okazać koniecznym osadzenie ich w kontekście wydarzeń społecznych, powiązanych z czynnikami gospodarczymi, kulturalnymi albo organizacyjnymi. Te czynniki zdają się być kluczowe w zrozumieniu jakiego rodzaju odpowiedzi na pytania "jak" i "dlaczego" spowodowały podjęcie konkretnych decyzji odnośnie bezpieczeństwa.

Wykrywanie ataków sieciowych jest jedną ze składowych Strategii Cyberbezpieczeństwa Rzeczypospolitej Polskiej na lata 2019-2024 [1]. Złożony system operacyjny podparty zakrojoną na szeroką skalę siecią złożoną z mnogich, heterogenicznych urządzeń rzadko kiedy jest całkowicie bezpieczny i wolny od niedoskonałości mogących wpływać na poziom bezpieczeństwa. Pomimo wszystkich wysiłków podejmowanych w trakcie projektowania i wdrażania krytycznych systemów informacyjnych, bardzo prawdopodobnym jest, że szkodliwe czynności podejmowane umyślnie i w złym

[1] Uchwała nr 125 Rady Ministrów z dnia 22 października 2019 r. W sprawie Strategii Cyberbezpieczeństwa Rzeczypospolitej Polskiej na lata 2019-2024

zamiarze są w stanie odnieść sukces, tym samym zagrażając poufności, dostępności lub integralności systemu w trakcie jego okresu żywotności. System wykrywania ataków sieciowych ma za zadanie wykrywać takie ataki przeciwko system komputerowym i sieciom. IDS walczy z utajonymi zagrożeniami nieustannie monitorując działający system i analizując zebrane dane tak, aby wykryć czy przypuszczono atak, czy nie. Kiedy mechanizm monitorujący wykryje, że miał miejsce atak (lub że atak właśnie trwa), wszczynany jest alarm.

Podobnie, jak w przypadku innych detektorów, faktyczna jakość ich działania może być zilustrowana poprzez dwa rodzaje występujących błędów. Są to: błąd fałszywie dodatni, ang. *False Positive (FP)* - nie przeprowadzono żadnego ataku, lecz detektor wszczął alarm, i fałszywie ujemny - *False Negative (FN)* - atak nastąpił, ale nie został wykryty. Kiedy wzrasta odsetek wyników fałszywie ujemnych, obniża się przydatność narzędzia wykrywającego ataki. W najgorszym przypadku, kiedy liczba fałszywie dodatnich wyników jest zbyt wysoka, duża liczba fałszywych alarmów powoduje, że korzystanie z niego może powodować skutek odwrotny od zamierzonego - operatora ignorującego wskazówki z detektora.

Analiza przeprowadzana przez IDS bazuje na lokalnych danych pochodzących z szeregu niezależnych sond rozmieszczonych w różnych miejscach monitorowanego, rozproszonego systemu. Każde z tych urządzeń obserwujących odpowiedzialne jest za monitorowanie małej, konkretnie określonej części całego systemu i przekazywanie co się tam dzieje. W zależności od tego, czy sonda monitoruje czynności konkretnej maszyny, czy czynności w konkretnym punkcie sieci komunikacji, IDSy są podzielone na trzy główne klasy, tj. na HIDS (ang. *host-based intrusion system*; hostowy system wykrywania ataków sieciowych), NIDS (ang. *network-based intrusion system*; sieciowy system wykrywania ataków sieciowych) i IDS hybrydowe. Niezależnie od obserwowanych celów (urządzenia obliczeniowe czy łącza komunikacyjne), przekazana informacja albo odpowiada nieprzetworzonej obserwacji dotyczącej działania (np. wystąpienie zdarzenia) albo jest wynikiem pierwszej analizy niskiego poziomu, która rozpoznaje podejrzane zachowanie na poziomie lokalnym (np. alarm niskiego poziomu). Lokalnie wygenerowana informacja jest zwykle przechowywana w rejestrach i dziennikach zdarzeń. Jest również wysyłana w wiadomościach do narzędzia analizującego, które centralizuje dane i przeprowadza dogłębniejszą analizę globalną: to działanie jest kluczową cechą SIEM (ang. *Security Information and Event Management*, Zarządzania Informacjami i Zdarzeniami Bezpieczeństwa). Warto zauważyć, że taka analiza na dwóch poziomach (lokalnym i globalnym) może być zastąpiona bardziej złożoną strukturą hierarchiczną

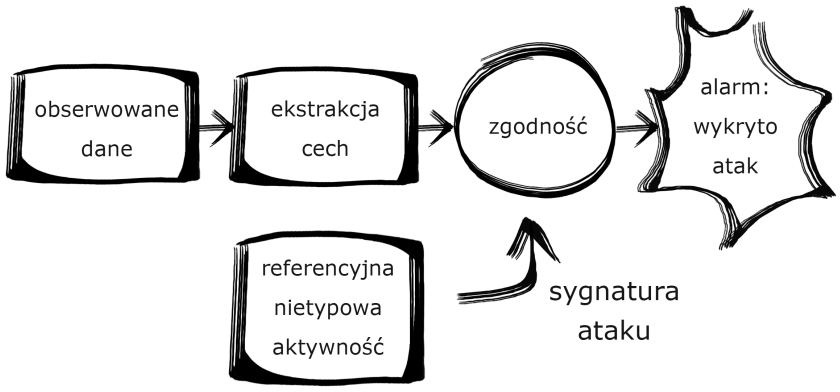
przeznaczoną do zbierania i analizy danych. Wszelako, zebrane informacje są uporządkowane według daty wystąpienia każdego elementu i w ten sposób tworzony jest niepowtarzalny przepływ (lub strumień) datowanych danych (zdarzeń i alarmów niskiego poziomu).

Dwa wspomniane podejścia do wykrywania ataków sieciowych są przedstawiane i omawiane oddzielnie: podejście bazujące na sygnaturach (ang. *signature-based approach*, sekcja 2.1) i podejście bazujące na anomaliach (ang. *anomaly-based approach*, sekcja 2.2).

Należy zauważyć, że niektóre wyzwania i trudności w wykrywaniu ataków sieciowych nie są zależne od wybranego podejścia. W szczególności jednym z istotniejszych wyzwań jest znalezienie odpowiedniego zbioru danych - elementu niezbędnego w procesie tworzenia metody wykrywania ataków sieciowych.

2.1. Podejście bazujące na sygnaturach

W podejściu opartym o sygnatury zgromadzone wzorce umyślnego, szkodliwego działania porównywane są do przychodzących danych i w chwili znalezienia pasującego odpowiednika zgłaszany jest alarm (rys. 3).



Rysunek 3. Podejście do wykrywania cyberataków oparte na sygnaturach

Podejście to stosowane jest już na poziomie urządzenia obserwującego, w celu wygenerowania alarmów niskiego poziomu. Może być również zastosowane po raz drugi, w trakcie scentralizowanej analizy, w celu uwzględnienia ataków wieloetapowych. W drugim przypadku, wykrywanie korelacji pomiędzy elementami przychodzącego strumienia danych ma na celu generowanie alarmów wysokiego stopnia. Wtedy ukierunkowane ataki nie

są takimi, które mogą być sprowadzone do pojedynczej akcji wykonanej przez atakującego na niepowtarzalnym urządzeniu. Podczas ataku wieloetapowego, atakujący wykonuje szereg szkodliwych czynności, niekoniecznie wszystkie na tym samym urządzeniu i niekoniecznie w jakiejkolwiek konkretnej kolejności [20]. Atak może być dokonany szybko lub być rozłożony w czasie. Złożony, wieloetapowy atak może być opisany przy pomocy rozmaitych technik. Z opisu ataku (zwanego sygnaturą) możliwe jest określenie w jaki sposób będzie można zaobserwować przypadek tego ataku, jeśli wpłynie on na konkretne, określone węzły systemu (tj. punkt widzenia obrońcy).

Aby zidentyfikować zarówno potencjalne cele atakującego jak i odpowiadające reakcje powiązanych sond, konieczna jest znajomość topologii sieci (lokalizacja urządzeń, łącza komunikacyjne) i narzędzi obserwacji (natura sond, lokalizacje, konfiguracje) [74].

Jako że odrębne instancje tego samego ataku mogą być wycelowane w różne zestawy maszyn, i jako, że sondy powiązane z tymi maszynami także mogą się różnić, pojedynczy opis ataku może odpowiadać różnym zestawom możliwych obserwacji. Ponieważ logiczne i czasowe ograniczenia czynności zwykle odpowiadają logicznym i czasowym ograniczeniom obserwacji, w odniesieniu do takich zestawów obserwacji używane jest słowo "wzorzec".

Reguły korelacji mają na celu zidentyfikowanie wszystkich wzorców, które mogą zostać znalezione w strumieniu: każdy wzorzec jest opisem możliwych reakcji obserwujących na konkretne wystąpienie ataku [73, 115].

Silnik korelacji to narzędzie analityczne które jako wkład pobiera strumień obserwacji oraz listę określonych reguł korelacji i sprawdza, dla każdego możliwego ataku, czy napastnik w tym momencie postępuje według odpowiadającej ścieżki ataku. Ręczne tworzenie reguł korelacji jest zadaniem mozolnym i wysoce podatnym na błędy. To wyjaśnia dlaczego w wielu stosowanych narzędziach brana pod uwagę jest tylko garstka prostych ataków. Najnowsze prace miały na celu zautomatyzowanie procesu tworzenia reguł korelacji [33].

Począwszy od opisu ataku (nierzadko dostarczonego przez specjalistów od bezpieczeństwa), automatyczna produkcja reguł korelacji wymaga posiadania bazy wiedzy zawierającej precyzyjny opis całego systemu. Główna trudność polega na tym, że należy ustanowić powiązanie pomiędzy jakąkolwiek czynnością określonego ataku i jego możliwym zaobserwowaniem ze strony rozmaitych sond. Taki proces odwzorowywania czasem okazuje się trywialny (w wypadku wykorzystywania dobrze znanej i opisanej luki w zabezpieczeniu), a czasem mniej oczywisty (na przykład w przypadku

mniej szkodliwej czynności, dla której opisy i klasyfikacje mogą różnić się w zależności od sondy).

Przewaga podejścia opartego o sygnatury polega na tym, że powoduje bardzo niewiele błędów fałszywie dodatnich (fałszywych alarmów) - jeżeli opis ataku i powiązane reguły korelacji są wystarczająco trafne. W podejściu opartym o sygnatury nowy atak lub atak który w stosunku do znanego został celowo zmieniony, niekoniecznie musi zostać wykryty (jest to tak zwany *0-day exploit*). Ponadto, wykrycie znanego ataku następuje tylko jeżeli istnieje wystarczająca liczba sond, które zostały rozmieszczone tak, by pokryć cały system, oraz zostały one poprawnie skonfigurowane.

Automatyczne generowanie reguł powoduje, że koszt pracy ludzkiej włożonej w tworzenie reguł maleje, więc liczba sygnatur ataków może w konsekwencji wzrosnąć. Wskutek tego silnik korelacji musi zmierzyć się z kwestią skalowalności, jako że może musieć zmagać się z tysiącami reguł korelacji równocześnie zachowując wysoką szybkość analizy [59].

Ponadto gdy zestaw reguł korelacji powiększa się, musi być częściej aktualizowany: niektóre urządzenia są usuwane lub przekonfigurowywane, inne dodawane, a przeciwdziałania podejmowane z przyczyn bezpieczeństwa (np. zamykanie portu) również wpływają na system. Restartowanie analizy od zera za każdym razem kiedy nastąpi dynamiczna aktualizacja jest ryzykownym rozwiązaniem: stare obserwacje (dokonane przed aktualizacją) będą ignorowane w trakcie analizy nowych reguł korelacji. W takim wypadku należy znaleźć kompromis pomiędzy kosztem częściowego ponownego zbadania poprzednich obserwacji i ryzykiem pominięcia pierwszych kroków nowych, określonych ataków.

2.2. Podejście bazujące na wykrywaniu anomalii

Podejście oparte na sygnaturach jest nieefektywne w wykrywaniu ataków, które są nowe, albo zostały celowo zmodyfikowane w celu ukrycia w wystarczającym stopniu, w ten sposób stając się tak zwanymi exploitami dnia zero [11]. Rozwiązanie tego problemu pojawia się w formie wykrywania anomalii. Poniżej opisana procedura może stanowić schematyczny szkic podejścia: najpierw należy ustalić wzorzec normalności (normalnego ruchu / czynności), następnie zestawić go z próbkami obecnego ruchu / aktywności. W wypadkach kiedy wzór odbiega od ustalonego modelu, generowany jest alarm (rys. 1).

Podejście to jest jednakże bardzo podatne na fałszywe alarmy. Często, gdy cechy charakterystyczne ruchu sieciowego (lub np. żądania HTTP w warstwie aplikacji) ewoluują, taka sytuacja interpretowana jest jako

anormalna, mimo iż jest to nieodłączna cecha użycia sieci i zachowania jej użytkowników [12].

W układach w których nowe ataki (lub nawet zmodyfikowane tylko rodziny złośliwego oprogramowania) pojawiają się nieprzerwanie, standardowe systemy ochrony stają się nieadekwatne dopóki, dopóty nie zostaną zebrane odpowiednie sygnatury [28].

Z drugiej strony, podejście bazujące na anomaliach (systemy które wykrywają anormalności w ruchu sieciowym, np. nietypowe żądania do baz danych) mają tendencję do dawania wyników fałszywie dodatnich (fałszywych alarmów) [6, 96].

Aby uzyskać model, który charakteryzuje normalny wzorzec, można zastosować rozmaite techniki powiązane z uczeniem maszynowym. Do najbardziej rozpowszechnionych należą algorytmy takie jak *Random Forest*(RF)(pol. Losowy Las), *Support Vector Machine* (SVM)(pol. Maszyna Wektorów Nośnych) lub techniki klastrowania takie jak *k-nearest neighbour* (k-NN) (pol. K Najbliższych Sąsiadów) lub *k-means* (pol. Algorytm Centroidów). W [101] została zastosowana metoda oparta na SVM w połączeniu z algorytmem fuzzy C-means clustering (pol. rozmyte klastrowanie C-średnich). Klastrowanie jest zastosowane do ekstrakcji cech, SVM natomiast wykonuje zadanie klasyfikujące. Interesujące podejście zaprezentowano w [63], gdzie autorzy tworzą znormalizowaną entropię sześciu cech opartych na przepływie sieciowym (NetFlow[2]) dla wykrywania anomalii a następnie zmieniają to z powrotem w zadania klasyfikacyjne dla ich hybrydowego modelu Particle-Swarm-Optimisation-SVM (pol. Optymizacja Rojem Cząsteczek - SVM).

Artykuł [94] dokonuje obszernej analizy Wykrywania Anomalii i Systemów Prewencji dla Smart Grids. Wśród nich, rozważanych jest 17 detektorów anomalii, włączając w to system oparty na klastrowaniu danych zebranych do honeypota - pułapki [122], detektor fałszywych danych polegający na ocenie czasoprzestrzennej [23], i detektor anomalii używający jednoklasowego SVM wyszkolonego na danych protokołów MMS i GOOSE [135].

W [32] k-NN użyty jest jako detektor anomalii (ang. *outlier detector*) stosowany w przepływie ruchu miejskiego. W [46] autorzy klastrują dane przepływu sieciowego w oparciu o technikę ruchomego okna.

W praktyce, projektując i tworząc inteligentne systemy dla wykrywania anomalii i cyberzagrożeń, można zauważyć, że w odniesieniu do cyberbezpieczeństwa i wykrywania ataków sieciowych, nie ma takiego indywidualnego klasyfikatora lub systemu IDS który umożliwiałby wykrycie wszystkich rodzajów ataków. Podobnie ten sam system (nawet jeśli był

[2] standard CISCO opisujący jednokierunkową sekwencje pakietów o wspólnym, konkretnym zestawie cech

uczony wykrywać ten sam rodzaj ataków) musi być ponownie wyuczony gdy nastąpią zmiany w monitorowanej sieci (topologia, usługi, właściwości, etc.).

2.3. Wykorzystanie metod uczenia maszynowego - przegląd literatury

Algorytmy wykrywania ataków sieciowych bazujące na sztucznej inteligencji stają głównie przed dwoma wyzwaniami. Tradycyjne algorytmy uczenia maszynowego (ML) są względnie szybkie, lecz równocześnie napotykają na wysoki odsetek wyników fałszywie dodatnich (FP). Z drugiej strony, Głębokie Uczenie (ang. *Deep Learning* - DL) wykazuje dużą precyzję, niskie wskaźniki FP, lecz uciążliwie długi czas obliczeniowy. Dlatego, autorzy [136] proponują rozwiązanie, które łączy zalety obu podejść. Zaproponowane rozwiązanie jest algorytmem monitorującym system operacyjny, korzystającym ze standardowego uczenia maszynowego jako z względnie szybkiego urządzenia monitorującego, nazywanego w artykule "*standard stage*", i kiedy następuje klasyfikacja, której status mieści się w kategorii "*borderline*" (pol. graniczny), inicjowany jest drugi etap algorytmu. Drugi etap, nazwany przez autorów "*uncertain*" (pol. niepewny), wykorzystuje Głębokie Uczenie do podjęcia ostatecznej, definitywnej decyzji czy proces jest złośliwy, czy nie.

Cyberbezpieczeństwo to szeroki temat, z rozlicznymi rozwiązaniami stworzonymi do odpierania różnych wektorów ataku [27]. Zastosowanie sztucznych sieci neuronowych (ang. *Artificial Neural Networks - ANN*) w celu wykrywania ataków sieciowych (*IDS*) oraz wykrywania złośliwego oprogramowania (ang. *malware*) nie jest właściwie nową koncepcją. Metody oparte o ANN jako wsparcie wykrywania anomalii i złośliwego oprogramowania badane są od wielu lat [97]. W [39] autorzy starają się odnieść do problemów nadmiernego dopasowania (przeuczenia/przetrenowania sieci - ang. *overfitting*), wysokiego zużycia pamięci oraz znacznych kosztów standardowych *IDS*/wykrywania złośliwego oprogramowania z jednokierunkową siecią ANN (ang. *feed-forward ANN*).

Autorzy utrzymują, że ich metoda osiąga zbliżone wyniki do procedur, które nie są oparte o sieci neuronowe, jednakże przy mniejszym koszcie obliczeniowym. Procedura przetestowana została na wzorcowym zbiorze danych KDD'99. Z artykułu można wyciągnąć następujący wniosek: im mniej danych, tym lepiej, z uwagi na czas jaki maszyna potrzebuje, aby je przeliczyć.

W artykule [34] oceniono przycinanie (ang. *pruning* lub *drop-out*) ANN jako część optymalizacji sieci. W zasadzie jest to usuwanie neuronów należących do warstwy wejściowej albo do warstw ukrytych. Procedura taka przyspiesza ANN, ponieważ zmniejsza się potrzebna liczba obliczeń. Autorzy [75] ocenili przydatność sztucznej sieci neuronowej w IDS i określili wyniki jako obiecujące.

W [108] zastosowano analizę głównych składowych (ang. *Principal Component Analysis* - PCA) jako ekstraktor cech, zanim podano dane do ANN, zamiast dostarczyć dane wejściowe prosto ze zbioru danych. Jak ilustruje artykuł, procedura taka znacząco zmniejsza zarówno wymagania dotyczące pamięci jak i potrzebny czas uczenia. Dwie oceniane metody wykazały zbliżone wyniki pod względem skuteczności("*accuracy*"). Takie wyniki sprawiają, że zastosowanie PCA jest w oczywisty sposób lepszą opcją. Wykorzystanie metody Kernel PCA poprawia czas uczenia się ANN, lecz zużywa o wiele więcej pamięci niż tradycyjna PCA. Wykorzystanie obu metod daje podobne wyniki skuteczności, w związku z czym autorzy [87] dochodzą do wniosku, że najbardziej pożądane efekty daje użycie mieszanki różnych algorytmów.

Autorzy artykułu [117] prowadzili badania nad zastosowaniem procesorów kart graficznych w celu przyspieszenia IDS bazujących na ANN. Poprawienie działania zostało dowiedzione.

Autorzy [109] oceniają sieć ANN o jednej ukrytej warstwie porównując jej działanie do algorytmów takich jak SVM, Naiwny Klasyfikator Bayesowski oraz C4.5. Sieć ANN osiąga podobne lub lepsze wyniki w wykrywaniu złośliwego oprogramowania, ale dzięki wykorzystanej w tych badaniach prostszej naturze trzywarstwowej konstrukcji ANN wymaga ona mniej obliczeń niż pozostałe testowane algorytmy. Eksperymenty przeprowadzone zostały na zbiorze danych NSL-KDD, który jest zbiorem wzorcowym, następcą KDD'99.

W ostatnich kilku latach zaproponowano wiele różnych konfiguracji sieci ANN do zastosowania w IDS. Autorzy [113] oceniają szereg odrębnych architektur ANN w celu sprawdzenia związków pomiędzy *precision*, *recall*, *accuracy* a stopniem złożoności sieci. Podsumowując, autorzy wyciągają wniosek, że bardziej złożone sieci są w stanie, do pewnego stopnia, poprawić badane wyniki wykrywania ataków. Pogłębianie architektury jest jednakże obarczone zwiększaniem wymogów obliczeniowych.

Szereg algorytmów został w [27] oceniony pod kątem modelowania wykrywania anomalii w żądaniach HTTP. W artykule [118] autorzy prezentują ocenę płytkich i głębokich architektur sieci neuronowych pod kątem przydatności w IDS. W czasie testów szeregu konfiguracji na zbiorze danych KDD'99, ta sprawująca się najlepiej - głęboka sieć o trzech warstwach

ukrytych - osiąga wyniki lepsze zarówno od innych architektur sieci neuronowych jak i od zestawu klasycznych algorytmów uczenia maszynowego. Opisana metoda osiąga skuteczność na poziomie 0.930, *precision* - 0.997, *recall* - 0.915 a *f1-score* - 0.955.

W artykule [50] konwolucyjna sieć neuronowa (ang. *Convolutional Neural Network* - CNN) zasugerowana jest w celu ekstrakcji i klasyfikacji cech. Architektura, składająca się z trzech warstw konwolucyjnych splecionych z trzema warstwami typu max-pooling, osiąga skuteczność na poziomie 99.23% w trakcie przeprowadzania eksperymentu na zbiorze danych KDD'99 [30].

W podobny sposób, architektura sieci CNN wykorzystująca 3 warstwy splotowe - jedną 1x1, jedną 3x3 i jedną 5x5, z normalizacjami pakietów po każdej warstwie (z wyjątkiem warstwy w pełni połączonej) sprawdzona zostaje w [134]. W celu wyeliminowania problemu znikających/ eksplodujących gradientów, na każdą warstwę nakłada się ograniczenie do normalnej dystrybucji $N(0, 1)$. Warstwy splotowe są ustawione tak, aby podążać za modelem "Incepcji" ("*Inception*"). Obcięta Funkcja Liniowa (*Rectified Linear Unit* - ReLU) została użyta jako funkcja aktywacji. Zaproponowana architektura osiąga skuteczność na poziomie 94,11%, uczona i testowana na zbiorze danych KDD'99 [134].

Artykuł [91] opisuje połączenie dwóch architektur sieci neuronowych w celu wykrywania ataków sieciowych: Deep Confidence Neural Network dla ekstrakcji cech i ANN dla klasyfikacji. Eksperymenty przeprowadzone są na zbiorze danych KDD'99 [30]. Sieć typu Deep Confidence jest wariantem Ograniczonej Maszyny Boltzmana (RBM, z ang. *Restricted Boltzman Machine*) [107], sieci neuronowej używanej do określenia dystrybucji prawdopodobieństw w jej zestawie danych wejściowych. Autorzy dowodzą wyników lepszych o niemal 10 procent w czterech testach w porównaniu do Principal Component Analysis (PCA) [90].

Niektóre badania biorą pod uwagę możliwość użycia sieci typu *Long Short-Term Memory* (LSTM) do wykrywania ataków sieciowych. LSTM zostało stworzone w celu rozpoznawania długoterminowych zależności w danych. W [5] sieć LSTM wykorzystuje optymizator ADAM (ang. *Adaptive Moment Estimation*). Autorzy uznają, że architektura LSTM połączona algorytmem ADAM nadaje się dla IDS.

W podobny sposób autorzy [51] testują konfigurację LSTM pod kątem jej przydatności dla IDS. Uczona jest tu również na zbiorze danych KDD'99. LSTM jest zoptymalizowana pod kątem pożądanych hiperparametrów i gdy zestawić ją z tradycyjnymi algorytmami Uczenia Maszynowego, wyniki są obiecujące.

Kilka artykułów dotyczy jednej z najbardziej nowatorskich konstrukcji sieci ANN, opartej o komórki GRU (z ang. *Gated Recurrent Unit*) [25] i jej użycia w systemach IDS [127] i [89], z wykorzystaniem zbiorów danych odpowiednio KDD'99 i NSL-KDD. GRU jest rozszerzeniem architektury sieci neuronowej o umiejętności rozpoznawania wzorców czasowych w danych, ale nie aż tak podatnym na przeuczenie jak wariant Long Short-Term Memory. Osiągnięta przez badaną sieć skuteczność przekracza 98% dla [89] i 99% dla wszystkich konfiguracji GRU w [127].

Artykuł [60] ocenia sposób w jaki PCA poprawia wyniki GRU dla systemów wykrywania ataków sieciowych, według autorów osiągając bardzo dobre wyniki.

Zastosowanie wariantu Gated Recurrent Units (E-GRU) oceniono w [42] w charakterze etapu wstępnej obróbki danych. Autorzy zauważają poprawę przewyższającą najbardziej nowoczesne metody na wzorcowym zbiorze danych - ISCX2012. Osiągnięta skuteczność sięga 99.9%, jednakże użycie pamięci przez E-GRU okazuje się być 32-krotnie wyższe niż w wypadku standardowego GRU.

Autorzy [128] podnoszą wyniki IDS stosując model głębokiej sieci neuronowej z automatyczną ekstrakcją cech. GRU jest użyta jako ekstraktor cech, który podaje dane wyjściowe do wielowarstwowego perceptronu (MLP), który następnie używa funkcji softmax dla uzyskania ostatecznej klasyfikacji. Eksperymenty przeprowadzone zostały na zbiorach danych KDD99 oraz NSL-KDD, osiągając skuteczność w wysokości 99.98% dla KDD99 i 99.55% w przypadku NSL-KDD.

W [81] znaleźć można przegląd 43 artykułów na temat zastosowania sieci ANN w IDS. Według autorów, badania w dziedzinie będą coraz częstsze i przewiduje się, że w nadchodzących latach stanie się to popularnym tematem.

W [131] oceniano wykrywanie złośliwego oprogramowania oparte o *NetFlow* przy użyciu konwolucyjnej sieci neuronowej. Autorzy sugerują, że współczesne metody wykrywania zbyt mocno polegają na wybranych polach pakietów, np. na numerze portu, co jest niejako martwym punktem dla złośliwego oprogramowania używającego nieprzewidywalnych numerów portów i protokołów. Zamiast tego zaproponowano 35 cech wyekstrahowanych z przepływu pakietów z danych pochodzących z projektu IPS Stratosphere. Wybrano po 2000 próbek danych w każdej z klas aby rozwiązać problem równowagi danych. Autorzy [131] wyekstrahowali 35 opartych o NetFlow statycznych cech do podania do konwolucyjnej sieci neuronowej i trzech innych algorytmów ML, mianowicie SVM, RF i wielowarstwowego

perceptronu (ang. *multi-layer perceptron*; MLP) dla ewaluacji. Architektura konwolucyjnej sieci neuronowej składała się z jednej warstwy wejściowej, pięciu warstw mapy cech, jednej spłaszczonej warstwy, dwóch warstw ukrytych i jednej warstwy wyjściowej. Autorzy wnioskują, iż algorytm RF przewyższa w działaniu pozostałe metody pod względem wszystkich trzech ocenianych metryk - *accuracy, specificity i sensitivity* (skuteczności, szczegółowości i czułości). Konwolucyjna sieć neuronowa była nieznacznie gorsza.

Autorzy [99] oceniają cały szereg algorytmów głębokiego uczenia przy użyciu wzorcowego zbioru danych z projektu Malicia. Oceniane procedury stosują cztery różne rodzaje metod ekstrakcji cech i używają dwie różne architektury autoenkodera - jedna warstwa i trzy warstwy. Sieć DNN ma *Exponential Linear Unit* w charakterze funkcji aktywacji we wszystkich warstwach oprócz wyjściowej, gdzie zastosowano funkcję sigmoidalną. Wykorzystany jest algorytm optymalizujący ADAM. Użyty algorytm RandomForest ma 100 drzew.

Przeprowadzane eksperymenty wskazują, że architektura dla najlepiej funkcjonującej głębokiej sieci neuronowej ma 7 warstw dla każdej metody doboru cech. Niemniej Random Forest w dalszym ciągu przewyższa w działaniu nawet najlepsze konfiguracje sieci. Autorzy wnioskują, iż użycie autokodera w połączeniu z głęboką siecią neuronową może mieć zbyt wielką moc jak na ten zbiór danych.

W [92] autorzy proponują nowatorski sposób wykrywania złośliwego oprogramowania typu "*stegomalware*", oparty na Sztucznym Systemie Immunologicznym (AIS; Artificial Immune System). Jest on zdolny wykrywać kod ukryty przy pomocy trzech powszechnie stosowanych narzędzi steganograficznych - F5, Outguess i Steghide.

Zaproponowana procedura ekstrahuje cechy z obrazów JPEG przy pomocy metody *Haar Wavelets*. AIS jest pewnego rodzaju samodefiniującym się systemem, który koncentruje się wokół kreowania, identyfikacji i utrzymywania własnego "ja", i pozbywania się wszystkiego co definicji tego "ja" nie odpowiada. Ten konkretny AIS oparty jest na Negative Selection. Szkolony system został porównany do metody opartej na SVM i wykazał lepszy wskaźnik wykrywania oraz zdecydowanie szybsze działanie.

Artykuł [4] podejmuje próby zastosowania konwolucyjnej rekurencyjnej sieci neuronowej w systemie HIDS (CRNN) w celu wykonania behawioralnej klasyfikacji złośliwego oprogramowania przy pomocy modelu uczonego na danych zebranych z plików preodczytu Microsoft Windows. Celem wykrywania behawioralnego jest zyskanie zdolności rozpoznawania rodzin złośliwego oprogramowania. Złośliwe oprogramowanie należące do konkretnej

rodziny wykazuje zbliżone cechy, co może zostać wykorzystane do stworzenia odrębnej sygnatury. Sygnatury mogą być utworzone z sekwencji kodu bajtowego, binarnych instrukcji assemblera lub zaimportowanej biblioteki DLL (Dynamic Link Library). Sygnatury dynamiczne mogą polegać na aktywności systemu plików, poleceń terminala, komunikacjach sieciowych lub sekwencji wywołania funkcji/systemu. Te sygnatury behawioralne są stosowane aby poszerzyć i uzupełnić sygnatury statyczne, które względnie łatwo podlegają obfuskacji. Sygnatury dynamiczne koncentrują się na zachowaniu złośliwego oprogramowania, co sprawia iż są w stanie rozpoznawać je pomimo obfuskacji. Metoda przedłożona przez autorów sprawdziła się na rzadkich rodzinach złośliwego oprogramowania o małej liczbie przykładów (małej próbce). Pierwsza warstwa sieci CRNN jest warstwą embedding, gdzie sekwencja nazw plików zasobu jest zmapowana do wektorów o dowolnych rozmiarach. Druga warstwa jest faktyczną warstwą konwolucyjną. Używa sekwencyjnego filtru, który skanuje listę nazw plików wyszukując lokalne powiązania pomiędzy wektorami embedding. Trzecia warstwa to warstwa grupująca typu max-pooling, która redukuje dane aby poprawić czas obliczeniowy i wyróżnić istotne reprezentacje danych. Czwarta warstwa to warstwa Bidirectional Long Short-Term Memory, która jest architekturą rekurencyjną. Ta warstwa uczy się długoterminowych zależności pomiędzy wektorami. Piąta warstwa to Global Max-Pooling a ostatnia - softmax. Autorzy donoszą iż ich model przewyższa w działaniu najnowocześniejsze metody.

W sieci typu CNN (konwolucyjna sieć neuronowa lub splotowa sieć neuronowa *ang. Convolutional Neural Network*) użytej w [82], 36 rodzin złośliwego oprogramowania (wraz z klasą nieszkodliwą) zostało użyte do wyuczenia sieci pod kątem ekstrahowania i oceny cech osiągając 85% trafności. To samo badanie uczy sieć CNN na zestawie 25 klas, używając odpowiednio zbiorów danych SARVAM i Maligm, osiągając 99% trafności. Autorzy korzystają z architektury VGG-16, głębokiej konwolucyjnej sieci neuronowej. Wektor cech zostaje umieszczony w macierzy i każdemu polu macierzy jest przypisana wartość od 0 do 255. To tworzy pewnego rodzaju obraz, po czym rozmiar obrazu jest zmieniony tak, aby dopasować go do wymagań VGG-16 - do formatu 244x244x3. VGG-16 ma warstwę konwolucyjną, warstwę zbierającą, warstwę normalizacyjną i warstwę fully-connected. Klasyfikator wyjściowy sieci CNN to albo Softmax, SVM albo k-NN z k=5. Autorzy wnioskują, iż do podniesienia wydajności klasyfikacji potrzebny jest większy zbiór danych.

Artykuł [79] ocenia trzy podejścia do głębokiego uczenia dla wykrywania złośliwego oprogramowania w Internecie Rzeczy (IoT; ang. *Internet of Things*). Są to trzy różne, konwolucyjne sieci neuronowe. Jako że ilość

złośliwego oprogramowania w IoT począwszy od 2015 roku podwaja się rokrocznie, wykrywanie algorytmów staje się kwestią zasadniczą. Autorzy podkreślają, że współczesne oprogramowanie antywirusowe wykrywa zaledwie 45% złośliwego oprogramowania.

Autorzy twierdzą, że w przeciwieństwie do złośliwego oprogramowania dla PC, złośliwe oprogramowanie dla IoT nierzadko nie stosuje żadnych technik obfuskacji, co sprawia iż łatwiej jest zarówno je rozłożyć na części jak i poddać technikom uczenia maszynowego.

Metoda nr 1 stosuje cechy sekwencji bajtów o stałym rozmiarze. Metoda nr 2 adaptuje cechy kolorowych obrazów stałego rozmiaru na konwolucyjnej sieci neuronowej AlexNet. Obraz uzyskiwany jest przy zastosowaniu krzywej Hilberta. Trzecia metoda korzysta z cech ekstrahowanych z sekwencji instrukcji assemblera. Oznacza to, że uczenie sieci CNN zajmuje więcej czasu.

Autorzy przygotowują dane dla sieci najpierw demontując wykonywalne pliki binarne aby stworzyć kody. Reprezentacja wektorowa jest skonstruowana w oparciu o techniki przetwarzania języka naturalnego (ang. *Natural Language Processing* - NLP), zakodowując każdy blok wartości do 30-wymiarowego wektoru o wartości rzeczywistej. Instrukcje są zsumowane pod kątem elementów. Zestaw filtrów zbudowany jest jako detektor cech, z ruchomym oknem wykrywającym lokalne powiązania w danych, w ten sposób tworząc mapę cech. Warstwa typu *max-pooling* jest użyta pomiędzy warstwami konwolucyjnymi i po ostatecznej konwolucji globalna warstwa grupująca użyta jest do skonstruowania ostatecznego wektoru.

Podejścia różnią się strukturami wyjściowych danych - jedne używają sekwencji bajtów, inne kolorowych obrazów, a jeszcze inne sekwencji instrukcji assemblera. Dwa pierwsze mają stałe rozmiary, trzeci ma rozmiar zmienny. Według autorów oba podejścia wydają się działać. Średnia skuteczność wynosiła 90.58 dla CNN_SEQ, 100 dla CNN_IMG oraz 100 dla CNN_ASM.

W artykule [98] znaleźć można metodę wykrywania złośliwego oprogramowania używającą dwuwymiarowego programu binarnego z cechami głębokiej sieci neuronowej. Jak donoszą autorzy, sieć osiąga zdatny do wykorzystania poziom wykrywania przy zdumiewająco niskim odsetku wyników fałszywie dodatnich - poziom wykrywania 95% a FP - 0.1%. Artykuł jasno przedstawia, że rozsądnym jest wyuczyć i stosować wysoce precyzyjny model ML względnie szybko. Struktura zastosowana w proponowanej metodzie obejmuje trzy komponenty, mianowicie ekstraktor cech, głęboką sieć neuronową (DNN) i "kalibrator wyniku", komponent który w zasadzie tłumaczy wyjściowe dane z sieci DNN na prawdopodobieństwo wykrycia złośliwego oprogramowania.

Dla uzyskania trafnego oznaczenia binaria zostały potraktowane narzędziem VirusTotal, które sprawdza pliki binarne za pomocą około 55 silników binarnych. Plik zostaje oznaczony jako złośliwy kiedy co najmniej 30% silników uzna go za złośliwe oprogramowanie.

Głęboka, sprzężona w przód sieć składająca się z 4 warstw wykorzystana jest w celu klasyfikacji; z trzema warstwami zawierającymi 1024-węzły, parametryczną rektyfikowaną linearną funkcję aktywacji (PReLU), dropout i funkcją sigmoidalną na ostatniej warstwie, z jednym węzłem. Eksperyment został przeprowadzony na zbiorze danych ponad 400 000 plików binarnych oprogramowania.

Artykuł [48] sugeruje system który działa osiągając skuteczność o wysokości sięgającej 95% na zbiorze danych posiadającym około 7000 próbek i jest znacząco szybszy do wyuczenia niż inne najnowocześniejsze metody. Autorzy budują sieć CNN zdolną nauczyć się cech charakterystycznych złośliwego oprogramowania z sekwencji instruktażowych. Metoda stosuje cechy niskiego poziomu uzyskane z plików wykonywalnych. Metoda jest zbudowana z dwóch komponentów - analizatora instrukcji i klasyfikatora: konwolucyjnej sieci neuronowej. Analizator instrukcji jest częścią, która zajmuje się wstępną obróbką danych. Instrukcje są zgrupowane według funkcjonalności i atrybucji aby ograniczyć rozmiar przestrzeni szkolenia. Sekwencje liczb grup instrukcji stają się danymi wejściowymi dla sieci CNN. Architektura sieci CNN pozwala na nienadzorowane uczenie się cech.

Każdy plik .exe konwertowany jest do pliku .asm, który składa się z instrukcji, a instrukcje z kolei składają się z kodów operacji operandów (argumentów). Sekwencje liczb grup są podane do sieci neuronowej. Dane wejściowe przechodzą przez warstwę embedding zanim podane zostaną do warstwy splotowej z funkcją aktywacji *Rectified Linear Unit* (ReLU). Po warstwie splotowej umieszczona jest warstwa typu max-pooling. Taki układ powtarzany może być wiele razy, a następnie ostatnia warstwa ukryta jest warstwą w pełni połączoną; po niej następuje softmax.

Układ ten zaimplementowany jest w bibliotece Torch [29] ze wsparciem akceleratora GPU. Taka konfiguracja jest znacząco lżejsza pod względem czasu obliczeniowego niż porównywalny algorytm Random Forest, nawet w układzie pięciowarstwowym.

W przypadku złośliwego oprogramowania na urządzenia mobilne, [129] ilustruje, że istnieje szeroki wachlarz czynności jakich złośliwe oprogramowanie może się podejmować. Fakt ten może spowodować, że wykrywanie mobilnego złośliwego oprogramowania okaże się ambitnym przedsięwzięciem. Niekończące się przystosowanie do nowych warunków zamienia próby prewencji w zdanie uciążliwe i żmudne. Jako nowatorskie rozwiązania autorzy proponują "*DeepRefiner*", algorytm głębokiego uczenia oparty na

semantyce, wykazujący skuteczność na poziomie 97,74%, przy wskaźniku wyników fałszywie dodatnich na poziomie 2,54%, zdecydowanie przewyższający w działaniu zestawione w porównaniu metody będące standardem w branży.

DeepRefiner przetwarza pliki .xml które zawarte są w plikach .apk aplikacji na Androida i rozszerza klasyfikację o semantykę kodu bajtowego. W pełni połączony perceptron wielowarstwowy (MLP, ang. *multilayer perceptron*) wykorzystany jest jako pierwsza warstwa wykrywająca, przetwarzając pliki .xml. W przypadku gdy nie jest w stanie zdecydować czy aplikacja jest nieszkodliwa, czy wręcz przeciwnie, przypisuje jej tymczasową etykietę "*niepewna*" i przesuwa na drugi poziom wykrywania, gdzie jest oceniana przy pomocy semantyki kodu. Powodem dla którego semantyka kodu działa tak skutecznie jest fakt, że zwykle stosowane metody obfuskacji, które polegają na ponownym ustawianiu klas, metod lub łańcuchów, nie wpływają na semantykę kodu.

W dodatku, bardzo wiele sekwencji kodu powtarza się w rozmaitych rodzajach złośliwego oprogramowania. *DeepRefiner* próbuje uchwycić sekwencje kodu zarówno na poziomie metody jak i aplikacji, łącząc skuteczność techniki Bytecode2Vec wraz z siecią Long Short-Term Memory Neural Network (LSTM). Deep Refiner jest w stanie wychwycić złośliwe sygnatury nawet jeśli wcale nie znajdują się obok siebie w kodzie źródłowym; wszystko to dzięki strukturze komórek pamięci LSTM.

Artykuł [5] ocenia zastosowanie sieci LSTM, rodzaju rekurencyjnej Sieci neuronowej (RNN) do wykrywania ataków sieciowych. Sieci LSTM są w stanie zapamiętywać wartości w ciągłych przedziałach czasowych. Dzięki temu są odpowiednie dla zastosowań IDS. Autorzy konstruują IDS oparty na sieci LSTM ze skutecznością wynoszącą 0.9997. Użyto zbioru danych CSIC 2010 HTTP.

Sieć LSTM jest udoskonaleniem architektury sieci RNN, zaprojektowanym aby być w stanie wychwycić długoterminowe zależności. W artykule, uczona jest na dwóch trzecich zbioru danych a następnie oceniona przy pomocy pozostałej jednej trzeciej. Wybrano algorytm optymalizujący ADAM. Autorzy uważają że architektura LSTM w połączeniu z algorytmem ADAM ma zastosowanie w IDS.

W [123] oceniany jest problem doboru odpowiednich cech dla IDS. Zaproponowany jest koncept hierarchicznego, czasoprzestrzennego systemu wykrywania ataków sieciowych opartego na cechach (HAST-IDS; z ang. *hierarchical spatial-temporal features-based intrusion detection system*). Architektura najpierw rozpoznawałaby cechy niskiego poziomu przy pomocy sieci CNN a następnie wysokiego poziomu cechy ruchu sieciowego przy

zastosowaniu sieci LSTM. Uczenie cech jest przeprowadzane automatycznie, w jego całokształcie. Ten proces obniża liczbę fałszywych alarmów. Aby przetestować ten pomysł, użyte są dwa wzorcowe zbiory danych: DARPA1998 oraz ISCX2012. Według autorów, w porównaniu do innych, najnowocześniejszych metod działanie takiego układu jest lepsze pod względem *accuracy, detection rate* i *false positive*.

Artykuł [120] bada sieć CNN w kombinacji z wieloma innymi algorytmami. Autorzy używają następującej, zapożyczonej z przetwarzania języka naturalnego architektury: warstwa konwolucyjna z warstwą grupującą, po niej warstwa w pełni połączona. Stosują ją tak jak i hybrydową wersję sieci CNN z innymi sieciami RNN na danych sieciowych reprezentowanych przez siatkę systematycznych interwałów czasowych. Warstwa konwolucyjna tworzy mapę cech przez przeprowadzanie konwolucji z filtrem na pakietach TCP/IP.

Użyty został łącznie milion czystych i złośliwych pakietów. W architekturach hybrydowych, wynik z sieci CNN jest podany dalej do sieci RNN, LSTM albo GRU. Rekurencyjna sieć neuronowa jest rozwinięciem wielowarstwowej sieci neuronowej, wprowadzającym cykliczną pętlę do wychwytywania wzorów ukrytych w odstępach czasu. Sieć LSTM jest udoskonaleniem sieci RNN, która wprowadza komórki pamięci zamiast cyklicznych pętli. Architektura sieci LSTM oferuje zdolność wychwytywania czasowych zależności z poprawioną rezyliencją na eksplodujące/znikające gradienty.

Sieć typu Gated Recurrent Unit (GRU) jest ewolucją sieci LSTM z mniejszą liczbą parametrów, co sprawia że jest mniej kosztowna obliczeniowo. Autorzy przeprowadzają swoje eksperymenty na wzorcowych zbiorach danych o długiej tradycji - KDDCup'99 i jego młodszym bracie, NSL-KDD. Po szeregu testów autorzy wysnuwają wniosek, iż architektura sieci CNN przewyższa w działaniu wyniki wyzwania KDD'99; jednakże żadna z konfiguracji hybrydowych które zaprojektowali autorzy nie sprostała poprawieniu tych wyników.

W artykule [119] autorzy szczegółowo testują głębokie sieci neuronowe pod kątem zastosowania IDS na wielu starych i nowych zbiorach danych. Przedstawiają również skalowalny, hybrydowy, oparty na sieci DNN model który nadaje się do stosowania w czasie rzeczywistym. Według autorów atak mógłby być usystematyzowany w pięciu częściach: rekonesans, eksploatacja, przybycie posiłków, zjednoczenie i w końcu grabież. Kiedy napastnik dociera do czwartego etapu, system jest w zasadzie otwarty.

Autorzy koncentrują się na przesłankach o samouczących się systemach, z nadzorowanymi, pół-nadzorowanymi i nienadzorowanymi algorytmami. Głównym problemem współczesnych rozwiązań uczenia maszynowego jest

ich wysoki wskaźnik wyników fałszywie dodatnich i ich wysoki koszt obliczeniowy. Autorzy przypisują tę wadę uczeniu się wzorów TCP/IP w sposób lokalny, w przeciwieństwie do rozwiązań Głębokiego Uczenia. Autorzy stosują szereg wzorcowych zbiorów danych: KDDCup 99, NSL-KDD, Kyoto, UNSW-NB15, WSN-DS, CICIDS2017.

W konfiguracji w tym eksperymencie użyte są Wielowarstwowy Perceptron (MLP) z pięcioma ukrytymi warstwami oraz funkcja aktywacji ReLU, z funkcją softmax albo funkcją sigmoidalną na warstwie wyjściowej. ReLU pomaga minimalizować problem znikających gradientów, jest to również najszybsza wśród funkcji aktywacji. Topologia sieci została znaleziona eksperymentalnie, konkretna konfiguracja wykazała się najdoskonalszym funkcjonowaniem na danych ze zbioru KDDCup'99. Sieć DNN sprawdziła się dobrze na wszystkich zbiorach danych, pokonując standardowe algorytmy ML (RandomForest i ADABoost osiągnęły podobne wyniki).

Artykuł [116] przedstawia hybrydowy system Snort IDS i oparte na NetFlow wykrywanie anomalii przy użyciu głębokiej sieci neuronowej - konkretnie *Stacked Autoencoder* - ze wskaźnikiem True Positive wyższym niż 90% i mniej niż 5% fałszywych alarmów. System osiąga również skuteczność na poziomie 85%, przy zaledwie 20 cechach pozyskanych z rzeczywistej sieci. Autorzy budują platformę testową *Software Defined Network* (SDN) w systemie Ubuntu i atakują konfigurację przy pomocy DDoS.

Dodatkowo, zwykły, nieszkodliwy ruch sieciowy jest podawany do konfiguracji. Oba typy ruchu sieciowego są podawane do konfiguracji przez hosty na maszynach wirtualnych. Pakiety są wychwycone przez SNORT (SNORT-Ryu oparty na SDN). Sieć DNN ma 3 ukryte warstwy z dziesięcioma, pięcioma i trzema neuronami. Model wprowadza wektor cech o rozmiarze 20. Klasyfikacja odbywa się w dwóch klasach - klasie nieszkodliwej i klasie anomalii.

W [95] przeprowadzona jest ewaluacja zarówno płytkich jak i głębokich sieci neuronowych, pod kątem ich wyników w dziedzinie wykrywania ataków sieciowych. Testy przeprowadzane są na zbiorze danych KDDCup'99. Przetestowane sieci DNN, które mają od jednej do pięciu ukrytych warstw oraz komplet klasycznych algorytmów uczenia maszynowego, wliczając w to *ADABoost*, *Decision Tree, K-Nearest Neighbour, Linear Regression, Naïve Bayes, Random Forest, SVM-linear* oraz *SVM-rbf.* Głębokie konfiguracje stosują kombinację warstw dense i dropout jedną po drugiej. Po porównaniu jasnym jest, że trzywarstwowa sieć DNN w działaniu na wzorcowym zbiorze danych KDD'99 przewyższa wszystkie pozostałe algorytmy i konfiguracje.

Artykuł [128] prezentuje model sieci głębokiej z automatyczną ekstrakcją cech, biorąc pod uwagę cechy charakterystyczne związane z czasem.

Metoda wykorzystuje *Gated Recurrent Unit* (GRU) w kombinacji z wielowarstwowym perceptronem i funkcją softmax. Eksperymenty przeprowadzone zostały na zbiorach danych KDD'99 i NSL-KDD; autorzy donoszą o wiodącym działaniu przy ogólnym wskaźniku wykrywania 99,42% na KDD99 i 99,31% na NSL-KDD, przy *False Positive Rate* odpowiednio 0,05% i 0,84%. Konfiguracja eksperymentalna ma układ kaskadowy składający się z warstwy wstępnej obróbki danych, warstwy GRU, MLP i warstwy wyjściowej. Warstwa GRU to dwukierunkowe GRU zastosowane jako ekstraktor cech; element nowatorski w tym artykule. Funkcja optymalizująca SGD jest użyta do uczenia sieci DNN a *cross-entropy* jako funkcja kosztu. Autorzy porównują trzy różne konfiguracje - GRU + MLP, LSTM + MLP oraz dwukierunkową rekurencyjną sieć neuronową (*BRNN - Bidirectional Recurrent Neural Network*).

Autorzy [77] przeprowadzają kolejne porównanie funkcjonowania architektur głębokiego uczenia na zbiorach danych NSL-KDD. Oceniane metody to autokodery (sparse, denoising, contractive, convolutional), sieci LSTM i CNN. Autoenkodery uczą się reprezentacji cech przez zmiany przestrzeni cech przy pomocy redukcji lub zmiany wymiarów. Wykorzystane Autoenkodery mają strukturę trzywarstwową: warstwę wejściową, ukrytą warstwę i warstwę wyjściową. Warstwy wejściowe i wyjściowe zawierają tę samą liczbę węzłów. Kiedy warstwa ukryta ma mniej neuronów niż warstwy wejściowe/wyjściowe, nazywana jest warstwą wąskiego gardła (ang. *bottleneck*)/różnicującą/kodującą/oddzielającą.

Wymuszenie wąskiego gardła sprawia, że autoenkoder pozyskuje najbardziej znaczące cechy. Wielowarstwowe konfiguracje są również możliwe dla autokoderów. Ważną cechą charakterystyczną autoenkodera typu *sparse* jest to, że zmniejsza prawdopodobieństwo przeuczenia. Autoenkoder odszumiający ("denoising") odzyskuje nienaruszoną wersję próbki przez minimalizację funkcji celu. Wyjątkową zaletą Autoenkoderów Kurczliwych (Contractive Autoencoders; ContAE) jest to, że mogą uczyć się reprezentacji w sposób który jest rezylientny na małe zmiany.

Kodery konwolucyjne (ConvAE) używają Spadku Stochastycznego Gradientu (ang. *Stochastic Gradient Descent* - SGD) do znalezienia nietrywialnych cech, które są dobrą inicjalizacją dla sieci CNN, unikając lokalnych minimów. W pełni połączona ConvAE, w przeciwieństwie do innych AE, zachowuje wyjątkowe położenie cech.

Metodologia stosowana w ewaluacji konfiguracji sieci DNN jest następująca: wszystkie architektury, za wyjątkiem ConvAE, są takie same. Autokodery są przygotowane przy pomocy podzbioru NSL-KDDTrain+. Warstwa wąskiego gardła zmniejsza przestrzeń cech do 16 wymiarów z 41.

Zredukowane reprezentacje stają się wkładem do MLP. Dla SparseAE L1 przeprowadzona jest regularyzacja poprzez wkład wąskiego gardła.

Dla DAE, wkład jest potraktowany szumem Gaussa, przy poziomie *corruption* wynoszącym 50%. W Kurczliwych AE funkcja kosztu jest zamieniona na kurczliwy koszt (*Contractive Loss*). ConvAE potrzebuje wkładu w formie obrazu. NSL-KDD jest skonwertowany na ciąg 32x32 2D. Ta metoda pozwala na odkrycie lokalizowanych powiązań w zbiorze danych. Wagi ConvAE są dzielone pomiędzy wszystkie lokacje macierzy wsadowej. Głębokie konfiguracje są porównywane z klasycznymi algorytmami ML - *Extreme Learning Machine, k-NN, Decision-Tree, Random Forest, SVM, Naïve-Bayes* oraz *Quadratic Discriminant Analysis*. Modele sieci DCNN i LSTM działają osiągając skuteczność na poziomie 85% oraz 89%, tym samym pokonując pozostałe badane konfiguracje.

2.4. Podsumowanie

Jak wynika z przeprowadzonej analizy literatury, zagadnienie wykrywania ataków sieciowych jest rozwiniętą dyscypliną z szeregiem istniejących rozwiązań. Należy jednak zwrócić uwagę na równie rozbudowany szereg nadal nierozwiązanych problemów.

Po pierwsze, zdecydowania większość badań, nawet tych wykorzystujących najnowsze zdobycze w dziedzinie uczenia maszynowego i głębokiego uczenia, stosuje do eksperymentów zbiór danych KDD'99 lub jego poprawioną odsłonę, NSL-KDD. Są to dane sprzed ponad dwudziestu lat i w żaden sposób nie oddają charakterystyki dzisiejszych problemów w dziedzinie. Brakuje natomiast badań nad adaptacjami metod uczenia maszynowego sprawdzającymi się w dzisiejszym ruchu sieciowym.

Po drugie, odnalezione w literaturze metody uczenia maszynowego charakteryzują się pewną dozą specjalizacji, która sprawia, że w zależności od zastosowania, osiągają lepsze lub gorsze wyniki. Przykładowo sieci z rodziny rekurencyjnych mają umiejętność odnajdywania zależności związanych z czasem. Jest to właściwość której brak większości pozostałych metod uczenia maszynowego. Jedną ze skutecznych metod pozwalających na skorzystanie z przewag różnych metod jest łączenie ich w schemat następujących po sobie procesów.

Po trzecie, w przeanalizowanej literaturze marginalną uwagę poświęca się kwestii optymalizacji hiperparametrów dobranych algorytmów. Czasem w artykułach znajduje się wręcz określenie, że użyte zostały domyślne ustawienia algorytmu. Odpowiedni dobór hiperparametrów może mieć krytyczne znaczenie dla osiąganych przez konkretną metodę wyników.

Po czwarte, stosunek liczby próbek poszczególnych klas w zbiorze danych ma wpływ na ostatecznie wyniki algorytmów uczenia maszynowego, jednak jest to temat rzadko poruszany w literaturze związanej z domeną cyberbezpieczeństwa. Może to być związane z faktem, że KDD'99 i NSL-KDD są zbiorami zbalansowanym. Nie zmienia to jednak faktu, że z natury rzeczy dane związane z wykrywaniem ataków sieciowych występujące w dzisiejszej rzeczywistości będą miały niezbalansowaną dystrybucję klas. Jest to problem, którego nie można pominąć w badaniu metod uczenia maszynowego do wykrywania ataków sieciowych.

3. Wykrywanie ataków w ruchu sieciowym - propozycja metody opartej o modyfikacje algorytmów uczenia maszynowego

Wcześniejsze działy tej pracy przedstawiły najnowocześniejsze rozwiązania w poruszanej dziedzinie; w poniższej części znajdą się metody i propozycje usprawnienień tych rozwiązań, poparte wynikami przedstawionymi w dziale następnym. Omówione zostaną zasady działania sztucznych sieci neuronowych, wraz z najnowszymi architekturami (Gated Recurrent Unit - GRU), omówiona będzie kwestia optymalizacji struktury i hiperparametrów sztucznych sieci neuronowych, wpływu problemu balansowania zbioru na wyniki osiągane przez model, jak i sposoby zaradzenia tej kwestii.

3.1. Proponowana metoda wykorzystująca sztuczne sieci neuronowe

Sztuczne sieci neuronowe (z ang. *Artificial Neural Networks - ANN*) są uniwersalnym narzędziem do modelowania. Ponieważ znalazły cały szereg skutecznych zastosowań, cieszą się akceptacją i renomą jako narzędzie do data miningu, klasyfikacji, regresji, klasteryzacji i analizy szeregów czasowych (ang. *time series analysis*).

Założeniem leżącym u podstawy ANN jest fakt imitowania, do pewnego stopnia, kompetencji w uczeniu się, które posiadają biologiczne sieci neuronowe, kładąc nacisk na właściwości sieci neuronowych odnajdywanych w ludzkich mózgach - jest to jednak imitacja o dużym stopniu abstrakcji [70].

Zaskakująca zdolność rozpoznawania wzorców przez ANN bierze się z jej elastyczności w dostosowywaniu się do danych. Ta rozbudowana wydolność w aproksymacji ma wyraźne znaczenie w obsłudze rzeczywistych danych, gdzie informacji jest dużo, ale wzory w nich występujące nadal pozostają ukryte i do ANN należy ich odnalezienie.

W ANN proces uczenia przeprowadzany jest poprzez aktualizacje wag następującymi po sobie pakietami próbek danych. Algorytm potrafi rozpoznać związki między zmiennymi, jak i również uogólniać w sposób, który pozwala na wysokie osiągi na nowych, wcześniej nie spotkanych przez model danych [31]. W najprostszych możliwych słowach jest to jak wykreślanie linii, płaszczyzny lub hiper-płaszczyzny przez zbiór danych [86].

Sztuczna sieć neuronowa o pojedynczej warstwie obliczeniowej nazywana jest perceptronem. Zbudowany jest on z warstwy wejściowej i warstwy wyjściowej, na której odbywają się stosowne obliczenia. Gdy próbki podane są do warstwy wejściowej, sieć przekazuje je do warstwy obliczeniowej.

Warstwa wejściowa zawiera d węzłów - neuronów, które odpowiadają za d cech $X = [x_1...x_d]$ oraz krawędź o wagach $W = [w_1...w_d]$. Neuron wyjściowy oblicza $W \cdot X = \sum_{i=1}^{d}(w_i x_i)$ [3]. W wypadku perceptronu przewidywana wartość jest binarna, wyznaczona przez znak wartości która jest wynikiem obliczeń na warstwie wyjściowej. W celu rozwiązania ewentualnego braku balansu w dystrybucji istnieje możliwość dodania wartości "*bias*".

Przewidywanie $\hat{y}$ jest wynikiem następującego równania:

$$\hat{y} = sign\{W \cdot X + b\} = sign\{\sum_{i=1}^{d} w_i x_i + b\} \tag{1}$$

Jak wynika z równania (1), *sign* jest funkcją aktywacji (ang. *activation function*), $\Phi(v)$. Sztuczne sieci neuronowe mogą wykorzystać szereg różnych funkcji aktywacji. W dzisiejszych głębokich sieciach w celu przyspieszenia procesu uczenia zazwyczaj wykorzystuje się *Rectified Linear Unit* (ReLU) albo "*Hard hyperbolic tangent*" (Hard Tanh) [3].

Błąd regresji może być wyrażony jako różnica między rzeczywistą wartością testową, a wartością przewidywaną, stąd $E(X) = y - \hat{y}$. Jeśli błąd nie jest równy 0, należy aktualizować wagi. Tak więc zadaniem perceptronu jest minimalizacja sumy kwadratów błędów między y oraz $\hat{y}$, dla wszystkich próbek w zbiorze D. Tak sformułowany cel nazywa się funkcją kosztu (2).

$$\sum_{(X,y)\in D} (y - sign\{W \cdot X\}) \tag{2}$$

Funkcja kosztu wyznaczana jest z uwzględnieniem całego zbioru X; wagi w są aktualizowane z "*learning rate*" α. Algorytm iteruje po całym zbiorze, aż dojdzie do optymalnych wyników [3]. Procedura ta nazywa się "*stochastic gradient descent*", wyrażana jak ukazano w równaniu (3)

$$W \Leftarrow w \ + \alpha E(X)X \tag{3}$$

Wielowarstwowa sztuczna sieć neuronowa tworzona jest poprzez dodawanie kolejnych warstw obliczeniowych, zwanych także ukrytymi. Sama nazwa wskazuje na nieprzejrzystą naturę tego rozwiązania (określane też jako "*black-box*"). Obliczenia wykonywane przez te warstwy są niewidoczne z perspektywy użytkownika. Próbki przekazywane są z warstwy wejściowej do kolejnych warstw z odpowiednimi obliczeniami dokonywanymi na każdym kroku, aż dojdą do warstwy wyjściowej.

Tak opisana procedura, w której dane wejściowe poruszają się tylko w jednym kierunku, nazywana jest siecią typu "*feedforward*" - sprzężoną w przód lub jednokierunkową [3]. Konkretna liczba neuronów w pierwszej ukrytej warstwie zazwyczaj nie przekracza liczby cech podawanych na warstwę wejściową. Sama ich liczba, tak jak i liczba warstw ukrytych, uzależniona jest wymaganej złożoności modelu, która z kolei wynika z danych [31].

Co prawda istnieją zastosowania, które wykorzystują w pełni połączone warstwy, jak i liczbę neuronów większą niż liczba neuronów w warstwie wejściowej, jednak wykorzystanie ukrytych warstw z liczbą neuronów mniejszą od liczby wejść pozwala uzyskać kontrolowaną redukcję w reprezentacji cech, która częstokroć poprawia wyniki sieci. Z dużym prawdopodobieństwem jest to rezultat kontrolowanego pozbycia się części szumu w danych [3].

Sieć neuronowa mająca za dużo neuronów może wykazywać niechciane zachowanie znane jako "*overfitting*", czyli nadmierne dopasowanie, przeuczenie lub przetrenowanie. Ten fenomen występuje gdy sztuczna sieć neuronowa dopasuje się do wzorców występujących konkretnym zbiorze danych na którym została wyuczona. Dopasowanie to jest tak ścisłe, że sieć ta może mieć problemy z działaniem na nowych danych, ponieważ aproksymacja nie jest wystarczająco ogólna [3].

W niniejszej pracy zbadano wpływ liczby ukrytych warstw, tak jak i wpływ liczby neuronów na tych warstwach i doboru odpowiednich hiperparametrów na wyniki osiągane przez sztuczne sieci neuronowe w kontekście cyberbezpieczeństwa.

3.1.1. Zastosowanie propagacji wstecznej błędu

Nauczenie perceptronu, który jest siecią o jednej warstwie obliczeniowej, jest dość nieskomplikowane - funkcja kosztu jest po prostu funkcją wag. Z sieciami zbudowanymi z wielu warstw procedura staje się bardziej zaawansowana, ponieważ warstwy wag wpływają na siebie nawzajem. Propagacja wsteczna (ang. *backpropagation*) oblicza gradient błędu jako sumę lokalnych gradientów względem wielu ścieżek do neuronu wyjściowego [3].

Algorytm ten składa się z dwóch faz. W pierwszej fazie dane podawane są do neuronów wejściowych i jeden po drugim obliczane są wyniki na kolejnych warstwach z obecnymi wagami. Wynikiem tej procedury jest predykcja, która jest porównywana do wyniku z próbki wykorzystanej do uczenia.

Faza wsteczna przekazuje wartość gradientu funkcji kosztu wszystkim wagom. Gradient użyty jest do aktualizacji wag, zaczynając od warstwy wyjściowej, krok po kroku po warstwach aż do warstwy pierwszej. Proces aktualizacji wag iteruje po wszystkich danych wykorzystywanych do uczenia. Każda iteracja nosi nazwę "epoch" czyli "epoki". ANN może potrzebować nawet tysięcy takich epok, by osiągnąć zbieżność.

3.1.2. Hiperparametry i poprawa wyników wybranych algorytmów poprzez ich optymalizację

Jedną z najważniejszych części architektury sztucznej sieci neuronowej jest funkcja aktywacji, gdyż jej wpływ na osiągane wyniki jest wprost bezpośredni.

W trakcie budowy modelu można wybierać z całej gamy funkcji aktywacji. Decyzja o jej wyborze jest decyzją dużej wagi zwłaszcza w architekturach wielopoziomowych, gdzie każda warstwa może mieć swoją własną, nieliniową funkcję aktywacji [3].

Każda osobna funkcja może mieć specjalny rodzaj wpływu na wyniki osiągane przez sieć ANN, tak samo jak na to czy sieć ANN w ogóle osiągnie zbieżność, jak i na ogólny charakter sieci.

Spośród szeregu funkcji aktywacji $\Phi(v)$ do testów wybrano następujące cztery:

- funkcja sigmoidalna (ang. *sigmoid - sig*)
- twarda funkcja sigmoidalna (ang. *hard sigmoid - hard sig*)
- tangens hiperboliczny (ang. *hyperbolic tangent - tanh)*
- obcięta funkcja liniowa (ang. *rectified linear unit - ReLU*)

Funkcja *Sigmoid* w dużym mierze pozwala na jednoznaczne predykcje, ponieważ X poniżej i powyżej pewnych wartości zbliża się do 0 lub 1. Dane wyjściowe z każdego neuronu są również znormalizowane w zakresie od 0 do 1. Jest jedną z najbardziej znanych funkcji aktywacji i dlatego zdecydowano o umieszczeniu jej w tych badaniach. Minusem funkcji Sigmoid jest podatność na problem znikającego gradientu. *Hard Sigmoid* jest wariantem funkcji Sigmoid dającym podobne korzyści, ale pozwalającym na szybsze uczenie sieci. Funkcja tanh wyparła funkcje sigmoidalne pozwalając na szybsze uczenie głębokich sieci, nie zdołała jednak rozwiązać problemu znikających gradientów. Dalsze badania doprowadziły do powstania funkcji ReLU, którą można spotkać w większości najnowszych architektur

głębokiego uczenia, jak *GoogleNet, ResNeXt, VGGNet, AlexNet* i inne [80]. Wybrano najpopularniejsze funkcje dla wyraźnego uwidocznienia wpływu wywieranego na konkretną architekturę.

Optymalne ustawienia sieci można znaleźć przy pomocy algorytmu grid search, który wykonuje kompleksową procedurę przeszukiwania przestrzeni hiperparametrów. Hiperparametry to parametry, które nie podlegają aktualizacji przez sam algorytm sieci neuronowej. Są to:

- liczba epok / epoch
- *batch size*
- funkcja aktywacji (ang. *activation function*)
- funkcja optymizująca (ang. *optimiser*)
- liczba ukrytych warstw, jeśli nie liczyć jej jako architekturę
- liczba neuronów na warstwach, jeśli nie liczyć jej jako architekturę

Gdzie liczba epok to liczba kompletnych cykli dostosowywania wag sieci, batch size to liczba próbek wykorzystana w jednej iteracji (epoce) [3]. Jest to wyczerpująca lista hiperparametrów dla użytych głębokich sieci neuronowych możliwa do optymalizacji przy użyciu metody grid search. Algorytm grid search może przetestować różne funkcje aktywacji, jak i różne optymalizatory. Optymalizatory, które wykorzystano w eksperymencie to:

- Adaptacyjny Estymator Momentu (ang. *adaptive moment estimation - ADAM*)
- Propagacja Pierwiastka Średnich Kwadratów (ang. *root mean square propagation - rmsprop*)
- Zrównoleglona Optymalizacja Stochastyczna (ang. *stochastic gradient descent - SGD*)

Wybrane jako najczęściej wykorzystywane optymalizatory w głębokim uczeniu, co określono na podstawie przedstawionego wcześniej badania literatury.

Przykładowe zestawienie wyników metody grid search można znaleźć w tab. 20.

3.1.3. Redukcja wymiarowości przestrzeni cech

Poszukiwanie wektora cech który wyrażałby naturę zbioru danych, ale bez potrzeby reprezentowania każdej cechy określa się mianem redukcji wymiarowości.

Procedura ta polega na konstrukcji n-wymiarowej projekcji, która jest wyrażeniem danych z przestrzeni k-wymiarowej. Pierwszymi korzyściami płynącymi z takiej procedury, jakie nasuwają się na myśl, jest zmniejszenie

wymaganej mocy obliczeniowej. Kolejną korzyścią jest zapobieganie "klątwie wielowymiarowości" (ang. *the curse of dimensionality*). Jest to problem który sprawia, że klasyfikatory oparte o uczenie maszynowe tracą na sprawności w miarę wzrostu liczby wymiarów [70]. Do zrównoważenia efektu wzrostu liczby wymiarów potrzebny jest wykładniczy wzrost liczby próbek.

Istnieje szereg podejść do redukcji liczby wymiarów. Ponieważ cechy, które nie poprawiają wyników modelu, można uznać za niepotrzebny szum, często implementowana jest selekcja cech. Procedura taka ma za zadanie dobrać te cechy, które mają największe zastosowanie w budowie modelu. Są też jednak inne metody ograniczania liczby cech, które dogłębnie wyrażają charakter zbioru danych.

Innym podejściem do zagadnienia jest Liniowa analiza dyskryminacyjna (ang. *Linear Discriminant Analysis* - LDA). Procedura ta ma dwa zadania - maksymalizację odległości między centroidami poszczególnych klas i minimalizowanie różnic między tymi klasami. Innymi słowy, LDA pomaga znaleźć granicę pomiędzy klastrami klas.

Inną znaną metodą redukcji liczby wymiarów jest "Analiza głównych składowych" (ang. *Principal Component Analysis* - PCA). PCA znajduje perspektywę w której wariancja danych jest maksymalna. Przykład podany w [70] pokazuje, że jeśli dane tworzą linię, przeprowadzenie PCA natychmiast uwidoczniłoby, że wariancja we wszystkich wymiarach, oprócz jednego, wynosi 0. W takim wypadku, skoro cechy na wszystkich tych wymiarach są bezużyteczne, można się ich pozbyć.

Zebrane dane częstokroć zawierają szum w wielu różnych cechach. Jeśli tylko szukany sygnał jest wystarczająco silniejszy od szumu, uzyskane projekcje zawierające maksymalną wariancję z dużym prawdopodobieństwem wyrażą esencję zebranych danych

3.2. Poprawa wyników algorytmów uczenia maszynowego przy pomocy balansowania zbioru danych

Tematem poniższych badań jest wpływ, który stosunek liczb próbek pośród różnych klas w użytym do uczenia zbiorze ma na wyniki osiągane przez algorytmy uczenia maszynowego wykorzystywane w domenie cyberbezpieczeństwa. Ogólnie rzecz ujmując, proces opartego o ML IDS można opisać krok po kroku w ten sposób: pakiet opisanych danych wykorzystuje się do nauczenia klasyfikatora; algorytm dopasowuje się do tych danych, tworząc model.

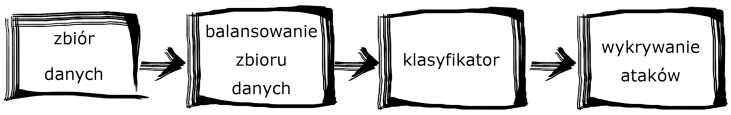
Następnym krokiem jest faza testowa, w której weryfikuje się osiągi uzyskanego modelu na testowym zbiorze danych, czyli pakiecie danych, który model widzi po raz pierwszy. W celu załagodzenia wpływu źle zbalansowanego zbioru danych na algorytm IDS dodaje się jeszcze jeden krok zanim zacznie się proces uczenia modelu (jak widać na rys. 6.1.3).

Sformułowanie problemu niezbalansowanych danych, (ang. *imbalanced data problem*), odnosi się do sytuacji, w której jedna lub więcej klas ma znacząco więcej próbek niż pozostałe klasy. Gdy taki stan ma miejsce, jego rezultatem częstokroć jest zła klasyfikacja próbek należących do klasy mniejszościowej. Takie zagrożenie dotyczy całego szeregu klasyfikatorów ML. Jest ono szczególnie dotkliwe w wypadku, gdy to właśnie wykrywanie próbek mniejszościowych ma największą wagę - jak w wypadku próbek nowotworów złośliwych, zdarzeniach będących oszustwami w finansach, albo, jak w wypadku niniejszej pracy, wykrywania ataków sieciowych. Dodatkowo, pogorszenie wyników klasyfikatora może przejść niezauważone jeśli jedyną metryką ewaluacyjną modelu jest skuteczność.

Powyżej opisana przypadłość uczenia maszynowego ma newralgiczne znaczenie w szeregu krytycznych sytuacji w życiu. W związku z czym zaproponowano szereg metod przeciwdziałania temu efektowi. Można je ogólnie skategoryzować w trzy podtypy: "*undersampling*", "*oversampling*" i metody "*cost-sensitive*". W niniejszej pracy zbadano szereg podejść do balansowania zbioru danych, jest podkreślony wpływ każdej z tych metod na zachowanie szeregu klasyfikatorów ML, po czym eksperymentalnie wyłoniono podejście najlepiej sprawdzające się w zadaniu wykrywania ataków sieciowych.

Głównym wkładem w dziedzinę i unikatową wartością tej części pracy jest uwydatnienie zagadnienia wpływu metod balansowania zbiorów danych na zachowanie klasyfikatorów uczenia maszynowego wykorzystywanych w dzisiejszym ruchu sieciowym, oraz faktu, że ten wpływ nie zawsze jest intuicyjny. Gruntownie przeewaluowano szereg metod balansowania zbiorów, a ich wpływ tak na zbiór danych, jak i na zachowanie klasyfikatorów jest dokładnie przedstawiony. Wszystko w kontekście praktycznego i istotnego zastosowania, jakim jest wykrywanie ataków sieciowych w nowoczesnych danych.

Widoczny na rys.6.1.3 bloczek odpowiadający uczeniu maszynowemu może zostać zrealizowany przez mnogość różnych metod uczenia maszynowego. Istotnie, najnowsze badania wykorzystują liczne nowe podejścia, włączając w to głębokie uczenie [79][82], [79][82], "*ensemble learning*" [133][56], najróżniejsze ulepszenia klasycznych algorytmów ML [57] itd. W niniejszej części pracy zdecydowano się wykorzystać trzy podstawowe algorytmy, by



Rysunek 4. Proces uczenia IDS opartego o uczenie maszynowe - z krokiem balansowania zbioru

położyć nacisk na działanie algorytmów balansowania zbioru. Wybrane metody to:

– Sztuczna sieć neuronowa - [105][108]
– Losowy Las (ang. *Random Forest - RF*) [13]
– naiwny klasyfikator bayesowski (ang. *Naïve Bayes - NB*) [70]

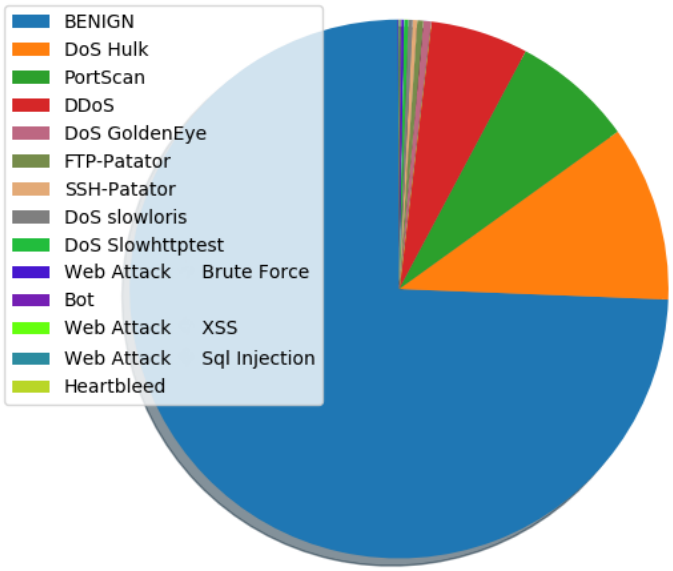
Reprezentują one trzy znacząco różniące się od siebie podejścia do uczenia maszynowego i wybrano je, by pokryć możliwie jak największy zakres efektów, które równoważenie zbioru może mieć na efektywność ML.

3.3. Metody balansowania zbiorów danych

W instancjach doznających problemu niezbalansowane danych liczba próbek wykorzystanych do uczenia, które należą do niektórych klas, jest drastycznie wyższa od liczb próbek w innych klasach

Problem niezrównoważonych danych był już badany pod pewnymi kątami w dziedzinie uczenia maszynowego i data miningu. W wielu przypadkach jest to kłopotliwa przypadłość, która wpływa na uczenie się algorytmów i w rezultacie pogarsza ich wyniki i ich efektywność w klasyfikacji [54]. Zazwyczaj w takich przypadkach klasyfikatory osiągają wyższą predykcyjną skuteczność w klasach większościowych i niższą predykcyjną skuteczność w klasach mniejszościowych. Rozwiązania tego problemu mogą dotyczyć bezpośrednio zbioru danych lub użytego algorytmu.

Nadchodzące akapity mają za zadanie przybliżyć zagadnienie balansowania zbiorów, przy nacisku na zastosowanie w cyberbezpieczeństwie, które jest jedną z rzeczywistych sytuacji zmagających się z tym wyzwaniem.



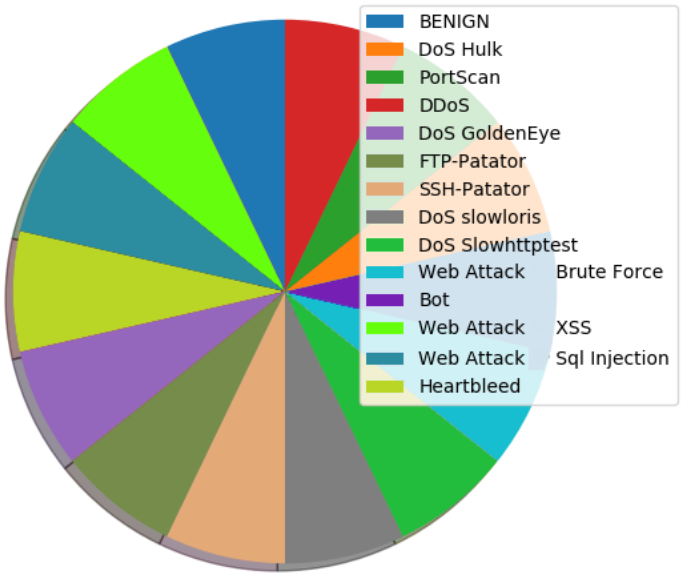
Rysunek 5. Dystrybucja klas w zbiorze CICIDS2017

3.3.1. Metody balansowania zbiorów wykorzystujące operacje na danych

Do tej kategorii należą dwie grupy technik, z których korzysta się w celu rozwiązania problemu niezbalansowanych danych. Obie grupy technik polegają na wytworzeniu nowego zbioru danych w oparciu o zbiór wejściowy. Realizuje się to poprzez różne podejścia do próbkowania. Te dwie ogólne grupy podejść to "*undersampling*" i "*oversampling*".

Metoda "*undersampling*" równoważy klasy w zbiorze poprzez zmniejszanie liczby próbek klas większościowych. Metodę tę wykorzystuje się gdy liczba rekordów w tych klasach jest na wystarczająco wysokim poziomie. Korzystając z tej metody można zachować niezmienioną liczbę próbek w klasach mniejszościowych i dobrać, korzystając z jednej z szeregu metod doboru, odpowiednią liczbę próbek z klasy większościowej.

W eksperymentach zawartych w tej pracy przetestowano szereg metod z kategorii *undersampling*, jedną z nich jest *Random Subsampling*. Efekt, jaki random subsampling ma na zbiór zilustrowany jest na rys. 7. Wpływ tej metody na wybrane algorytmy uczenia maszynowego odnaleźć można w tab. 5. Metoda ta polega na losowym doborze próbek z klas większościowych do założonej liczby.

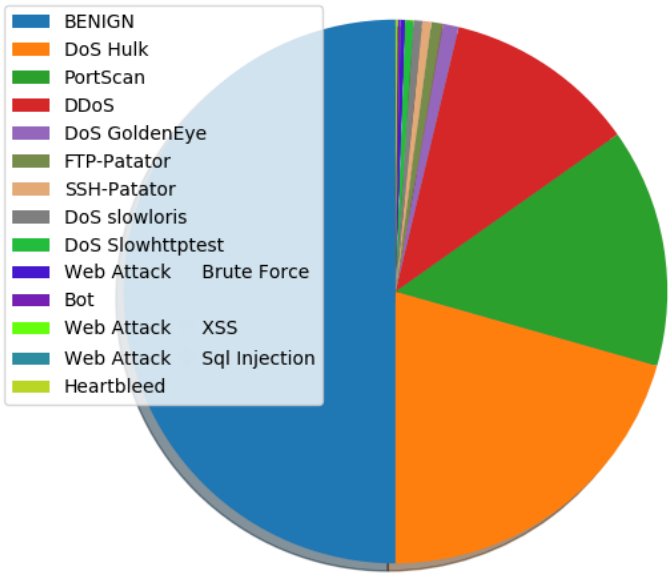


Rysunek 6. Rozkład klas w zbiorze CICIDS2017 po procedurze SMOTE

Istnieją podejścia, które wprowadzają do procesu próbkowania pewne heurystyki. Algorytm nazwany **NearMiss** [137] jest jednym z nich. Podejście to wykorzystuje algorytm *nearest neighbours* (pol. K najbliższych sąsiadów) do analizy i wyboru próbek z zbiorze. Algorytm *NearMiss* ma trzy różne formy; opcja wykorzystana w tej pracy wybiera te próbki, których średnia odległość od najbliższych próbek przeciwnej klasy jest najmniejsza. Efekt tego algorytmu na zbiór danych jest zilustrowany na rys. 8, a osiągnięte wyniki umieszczono w tab. 6.

Innym przykładem algorytmu z kategorii *undersampling* jest metoda **TomekLinks** [114]. Procedura wykonuje *undersampling* przez usuwanie tzw. Tomek-linków. Tomek-link istnieje gdy dwie próbki są swoimi najbliższymi sąsiadami. Dokładniej rzecz ujmując, Tomek-link pomiędzy próbkami z dwóch różnych klas x oraz y zdefiniowany jest jako $d(x, y) < d(x, z)$ i $d(x, y) < d(y, z)$ dla każdej próbki z. Zastosowana metoda usuwa próbki z klasy większościowej gdy jest ona częścią pary tworzącej Tomek-linki. Efekt tej metody na zbiór danych zilustrowany jest na rys. 9, efekt jaki ma na wyniki modelu można znaleźć w tab. 7.

Inne podejście do undersamplingu zakłada wykorzystanie centroidów uzyskanych przez zaaplikowanie metody klasteryzacji. W tym rodzaju algorytmów próbki należące do klasy większościowej są najpierw grupowane

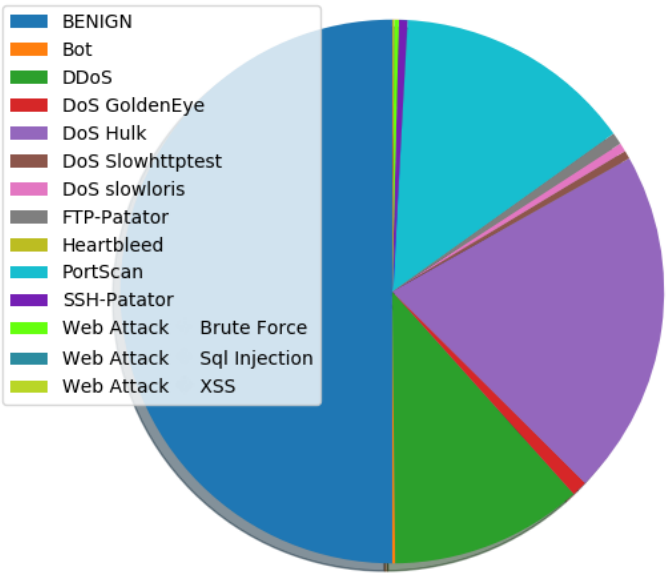


Rysunek 7. Rozkład klas w zbiorze CICIDS 2017 - Klasa "BENIGN" po procedurze Random Subsampling

(np. przy użyciu algorytmu centroidów - ang. *k-means*) i zastąpione przez uzyskane centroidy klastrów. W eksperymentach to podejście jest opisane jako **Cluster Centroids**. Wynik wpływu tej metody na zbiór danych zilustrowany jest na rys. 10 a na wyuczone modele ML w tab. 8

Z drugiej strony istnieją metody z grupy "*oversampling*". Stosuje się je gdy rozmiar pierwotnego zbioru danych jest względnie mały. W tym podejściu w celu uzyskania lepszego balansu pomiędzy klasami powiększa się liczbę wystąpień próbek w klasach mniejszościowych. Można to uzyskać wykorzystując techniki takie jak bootstrapping. W takim wypadku klasa mniejszościowa jest próbkowana z powtórzeniami. Innym rozwiązaniem jest wykorzystanie techniki **SMOTE** - (ang. *Synthetic Minority Oversampling Technique*) Technika Syntetycznego Oversamplingu Mniejszości [40].

Istnieją różne warianty i modyfikacje podstawowej wersji algorytmu SMOTE. Algorytm wybrany to eksperymentów w tej pracy to **Borderline SMOTE**. W tym podejściu próbki reprezentujące klasę mniejszościową są w pierwszej kolejności kategoryzowane na grupy: niebezpieczna, bezpieczna i szum (ang. *danger, safe, noise*). Próbka x jest uważana za należącą do kategorii *noise* gdy wszystkie próbki będące jej najbliższymi sąsiadami (wg. algorytmu *nearest-neighbours*) są przedstawicielami innej klasy niż



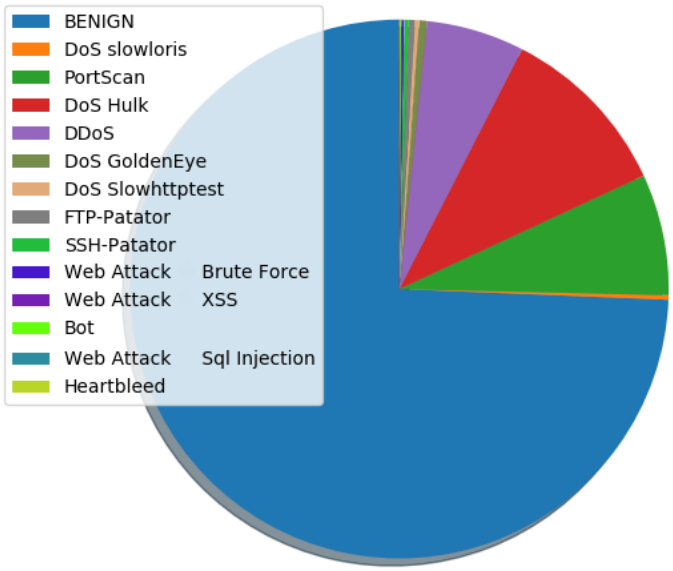
Rysunek 8. Rozkład klas w zbiorze CICIDS2017 po procedurze NearMiss

analizowana próbka x, *danger* gdy połowa jej najbliższych sąsiadów należy do innej klasy, *safe* gdy wszyscy jej najbliżsi sąsiedzi należą do tej samej klasy. W algorytmie Borderline SMOTE jedynie próbki z grupy *safe* wybierane są w celu oversamplingu [41]. Efekt jaki ta procedura ma na zbiór danych uwidoczniony został na rys. 6. Wpływ Borderline SMOTE na wyniki algorytmów ML można znaleźć w tab.9

Różne zastosowania omówionych metod na różnych zbiorach danych prowadzą do różnych wyników, przez co nie można przedłożyć jednej metody nad drugą, jako wyraźnie lepszą. Dla jasnego zobrazowania procesów zachodzących przy korzystaniu z metod wpływających na zbiór danych umieszczono pierwotny rozkład danych w CICIDS2017 na rys.5, rezultaty uzyskane przez wybrane algorytmy ML na niezbalansowanych danych znajdują się w tab.3.

3.3.2. Metody balansowania zbiorów wykorzystujące modyfikacje proponowanych algorytmów

Wykorzystywanie nieodpowiednich metryk ewaluacyjnych dla klasyfikatora wyuczonego przy pomocy niezbalansowanego zbioru danych może



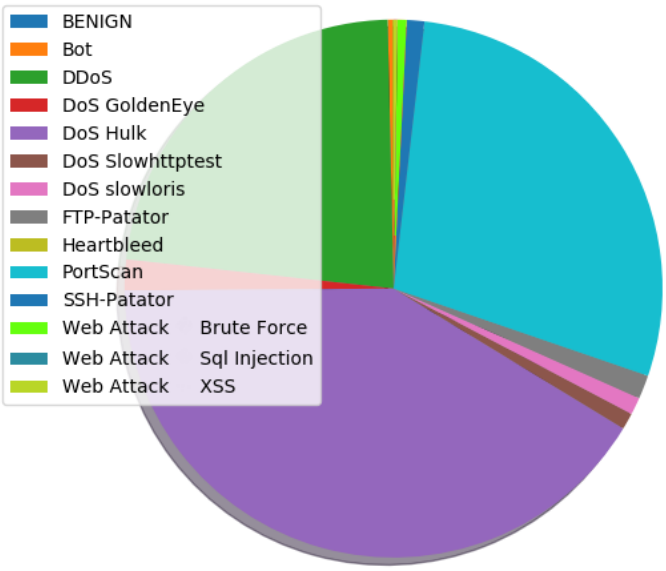
Rysunek 9. Rozkład klas w zbiorze CICIDS2017 po oczyszczeniu linków typu *Tomek*

prowadzić do podjęcia błędnych konkluzji na temat jego efektywności. Ponieważ większość algorytmów ML nie funkcjonuje najlepiej z niezbalansowanymi danymi, najczęściej występującym efektem ubocznym jaki można zaobserwować jest fakt, że klasyfikator ignoruje klasę mniejszościową.

Dzieje się tak, ponieważ klasyfikator nie jest wystarczająco penalizowany za błędną klasyfikację danych należących do klasy mniejszościowej. Z tego właśnie powodu zaproponowane zostały metody przeciwdziałania niezbalansowanym danym oparte o modyfikacje algorytmów, jako ulepszenie procedury uczenia.

Jedną z technik jest używanie innych metryk ewaluujących osiągnięcia algorytmu. Alternatywnymi metrykami odpowiednimi dla niezbalansowanych danych są:

- *precision* - wskazująca na procent trafnych próbek które zostały wyłonione przez klasyfikator
- *recall* (lub *sensitivity*) - wskazująca na całkowity procent wszystkich trafnych wystąpień, które zostały wykryte
- *f1-score* - obliczana jako średnia harmoniczna *precision* i *recall*



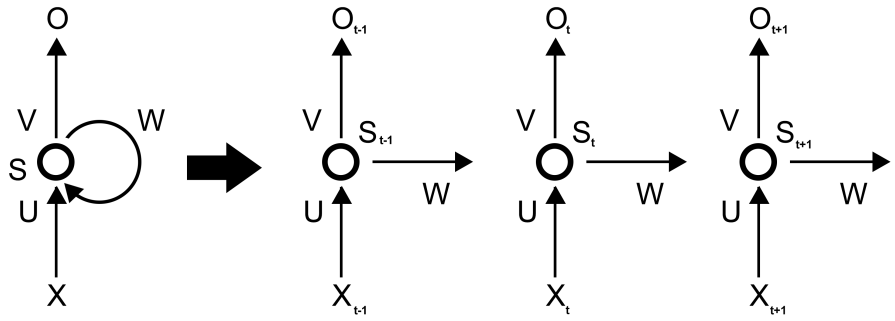
Rysunek 10. Rozkład klas w zbiorze CICIDS2017 po wykonaniu procedury próbkowania metodą Cluster-Centers

Inną techniką wykorzystywaną w dziedzinie jest klasyfikacja typu "*cost-sensitive*". Najnowsze badania pokazują jej efektywne działanie w rozwiązywaniu problemu zbalansowania klas w zbiorach danych o wielkiej skali. Formułę na klasyfikację typu "*cost-sensitive*" można zdefiniować w sposób uwidoczniony na równaniu (4):

$$\hat{\theta} = \min_{\theta} \left\{ \frac{1}{2}||\theta||^2 + \frac{1}{2}\sum_{i=1}^{N} C_i||e_i||^2 \right\} \tag{4}$$

gdzie θ wskazuje parametry klasyfikatora, e_i jest błędem klasyfikatora w odniesieniu do i-tej próbki (z N) oraz C_i jest miarą ważności i-tej próbki.

Ideą stojąca za metodą "*cost-sensitive*" jest nadanie wyższego stopnia ważności C_i próbkom z klasy mniejszościowej, by zredukować błąd kierujący skłonnością klasyfikatora w stronę klasy większościowej. Innymi słowy, tworzy się funkcję kosztu, która bardziej penalizuje nieprawidłową klasyfikację klasy mniejszościowej w stosunku do penalizacji błędnej klasyfikacji klasy większościowej. Najbardziej obiecujące wyniki metody "cost-sensitive" osiągnął w przeprowadzonych badaniach algorytm Random Forest, jego osiągi zostały zebrane w tab. 12.



Rysunek 11. Rozwinięta struktura rekurencyjnej sieci neuronowej [37]

3.4. Propozycja metody wykorzystującej architekturę sieci neuronowej typu Gated Recurrent Unit

Wedle najlepszej wiedzy autora w chwili pisania niniejszej pracy architektura *Gated Recurrent Unit* nie jest jeszcze gruntownie przebadana w kontekście wykrywania ataków sieciowych od czasu jej powstania w 2014 [25].

3.4.1. Rekurencyjne sieci neuronowe

Standardowe architektury sztucznych sieci neuronowych nie radzą sobie najlepiej w sytuacjach gdy potrzebna jest umiejętność rozpoznawania zależności między danymi rozłożonymi w sekwencji. Z tego powodu nie osiągają oczekiwanych wyników na danych, które progresywnie budują relacje z wcześniejszymi instancjami, jak szeregi czasowe, albo tekst [14].

By sieć uporała się z takim rodzajem danych, potrzebny jest inny rodzaj architektury. Odpowiedzią na to wyzwanie jest rekurencyjna sieć neuronowa (ang. *Recurrent Neural Network* - RNN); architektura która zawiera w sobie sprzężenie zwrotne. RNN jest w stanie uczyć się bez opierania się na założeniu Markova, które domniema, że przy założonym teraźniejszym stanie, żadne kolejne stany nie są w żaden sposób zależne od stanów przeszłych [106]. Dane wyjściowe o indeksie czasowym T aplikowane są jako jedno z wejść do indeksu czasowego T+1 lub z powrotem same do siebie [37]. Rozwinięta struktura RNN ukazana została na rys. 11.

Taki sposób propagowania wag przez czas sprawia, że architektura RNN zmaga się ze swoim własnym problemem. Wspomniane wartości są mnożone rekursywnie, przez co gdy wagi są zbyt małe, kolejne procesy sprawiają, że stają się progresywnie coraz mniejsze. Gdy wagi są duże, zaczynają gwałtownie przyrastać, a wartości wyjściowe idą w kierunku nieskończoności. Sytuacja taka nazywa się problemem znikającego gradientu, lub odpowiednio problemem eksplodującego gradientu (ang. *vanishing-, exploding gradient problem*) [37].

3.4.2. Long Short-Term Memory i Gated Recurrent Unit

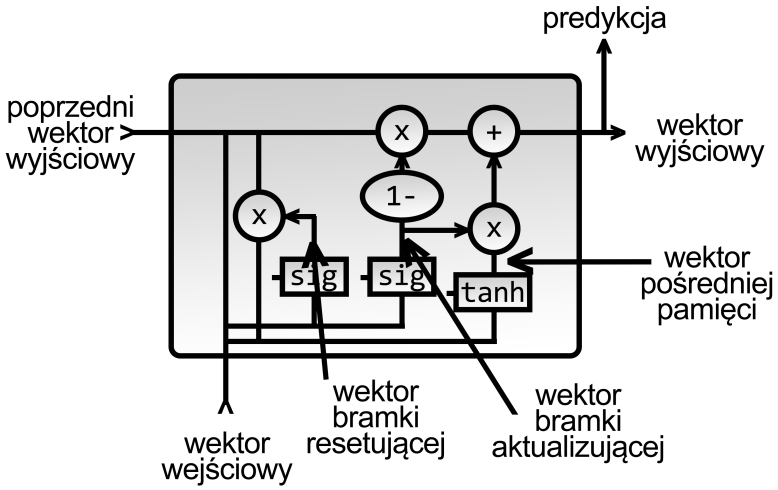
Problem znikających/eksplodujących gradientów powstaje, gdy macierze są wielokrotnie mnożone, destabilizując wynik. Ma to miejsce, gdy RNN musi przetworzyć wystarczająco długą sekwencję. RNN nie doświadcza tego problemu dla krótkich sekwencji.

Wynika z tego, że RNN ma pewną pamięć krótkoterminową, ale osiąga wyniki gorsze od spodziewanych kiedy potrzebna jest pamięć długoterminowa. W celu uporania się z tym problemem zaproponowano sieć *Long Short-Term Memory* (LSTM). Sieć ta wykorzystuje właściwość nazwaną "*cell-state*" (pol. stan komórki), w celu zachowania fragmentów pamięci długoterminowej.

Gated Recurrent Unit (GRU) [25] można postrzegać jako uproszczenie architektury LSTM, u którego działania leży ta sama podstawowa zasada, ale nie do końca przenosząca wszystkie właściwości. GRU używa *"reset gate*" (pol. bramki resetującej) w celu częściowego resetowania ukrytych stanów sieci [14], struktura komórki GRU została zilustrowana na rys. 12. Modele stworzone przy pomocy GRU zazwyczaj mają mniejszą złożoność od modeli stworzonych przy pomocy LSTM [37]. GRU jest jednak bardziej wydajne niż LSTM [14]. W ramach badań do niniejszej pracy przetestowano architekturę GRU na danych ze zbiorów NSL-KDD i CICIDS2017.

3.5. Propozycja metody opartej o architekturę Gated Recurrent Unit i algorytm Random Forest w dwóch opcjach balansowania danych

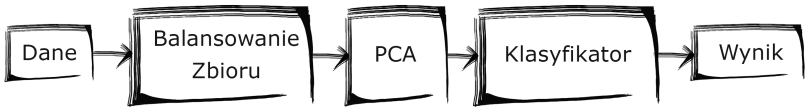
We wcześniejszych sekcjach zaproponowano szereg możliwych modyfikacji algorytmów uczenia maszynowego, w następnym rozdziale znajdują się osiągnięte przez zmodyfikowane algorytmy wyniki.



Rysunek 12. Struktura komórki Gated Recurrent Unit [25]

Każda z omówionych metod ma swoje mocne i słabe strony. Wpływ metod balansowania danych na zachowanie algorytmów uczenia maszynowego nie zawsze jest oczywisty, ale dobrze dobrana metoda pomaga uzyskać lepsze osiągi dla klas mniejszościowych. Odpowiednie ustawienia hiperparametrów potrafią drastycznie zmienić osiągi tej samej architektury. Zaprezentowana wcześniej sieć Gated Recurrent Unit potrafi znaleźć zależności czasowe występujące pomiędzy próbkami danych i jej zastosowanie jest bardzo obiecujące, ale, jak pokazuje eksperyment, ma tendencję do gubienia klas mniejszościowych nawet pomimo użycia metod balansowania danych. Jednym z najbardziej prominentnych algorytmów uczenia maszynowego, również w wypadku zbioru danych CICIDS2017 okazał się Random Forest, zwłaszcza wyposażony w zdolność balansowania danych metodą cost-sensitive. Jest to jednak algorytm podatny na szum i przeuczenie.

Jedną z propozycji wykorzystania przewag jednych metod do zrównoważenia słabych stron innych metod i nawzajem jest ustawienie kolejnych omówionych wcześniej technik w schemat następujących po sobie procesów. Po wczytaniu danych i standardowych zabiegach preprocessingu, należy zadbać o zbalansowanie zbioru. Następnie, po skalowaniu, można wykorzystać PCA w celu redukcji wymiarowości. Uzyskane komponenty podać na klasyfikator oparty o GRU, DNN, RF lub inny klasyfikator osiągający dobre wyniki na bieżących danych, tak jak ukazano na rys. 13



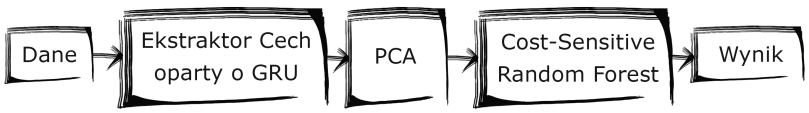
Rysunek 13. Ogólny schemat metody - propozycja pierwsza

Rysunek 14. Ogólny schemat metody - propozycja druga, wersja A

Druga propozycja modyfikacji metod opartych o uczenie maszynowe polega na wykorzystaniu tylko unikatowych cech konkretnych metod i złączeniu ich w jedno, "*hybrydowe*", podejście. By osiągnąć taki rezultat można wykorzystać sieć opartą na architekturze GRU w roli ekstraktora cech, po czym zastosować PCA w celu ograniczenia dalszego przetwarzania tylko do cech wyrażających odpowiednią wariancję, następnie podać tak uzyskane komponenty na algorytm random forest balansujący zbiór danych w sposób "cost-sensitive" (lub zbalansować dane przed podaniem na GRU). Propozycje te zilustrowano na rys. 14 i rys. 15.

3.6. Podsumowanie

Powyższy rozdział stanowi przedstawienie znanych jak i nowych metod opierających się o uczenie maszynowe oraz propozycji nowych technik



Rysunek 15. Ogólny schemat metody - propozycja druga, wersja B

pozwalających na poprawienie osiągów algorytmów uczenia maszynowego. W następnym dziale przedstawione zostaną wyniki zastosowania tych metod na nowoczesnym ruchu, zawierającym odpowiednio świeże próbki ataków. W dalszej części pracy przeprowadzona zostanie również ewaluacja zastosowania metody opartej na architekturze głębokiego uczenia GRU w zagadnieniu wykrywania ataków sieciowych w oparciu o wzorcowy zbiór danych, jak i odpowiednio nowy zbiór ataków sieciowych, posiadający zapis świeżego ruchu podkładkowego, jak i przykłady zagrożeń ważnych dla dnia dzisiejszego.

4. Ewaluacja skuteczności zaproponowanych rozwiązań

Wcześniejsze sekcje niniejszej pracy przedstawiły zasady działania sztucznych sieci neuronowych, wraz z najnowszymi architekturami (GRU), omówiono w nich kwestię optymalizacji sztucznych sieci neuronowych, wpływu problemu balansowania zbioru na wyniki osiągane przez model, jak i sposoby zaradzenia tej kwestii. W poniższym dziale odnaleźć można wyniki związane z badaniami we wszystkich tych zagadnieniach, wraz z porównaniem do najnowocześniejszych metod na wzorcowych zbiorach danych.

4.1. Opis wybranych zbiorów danych

4.1.1. Zbiór odkrywania wiedzy w bazach danych laboratorium bezpieczeństwa sieciowego - NSL-KDD

NSL-KDD [112] jest ulepszoną wersją poprzedniego uznanego w branży zbioru - KDD'99. NSL-KDD powstał jako próba naprawienia problemów występujących w KDD'99, które wielokrotnie podnoszono w literaturze branżowej. Pomimo że zbiór ten jest, niestety, raczej stary i nadal ma kilka nieszczęśliwych cech, brak ogólnodostępnych zbiorów w dziedzinie IDS oraz jego wypracowany przez lata status zbioru wzorcowego (ang. *benchmark*) sprawia, że nadal jest jednym z najczęściej wybieranych zbiorów w dziedzinie cyberbezpieczeństwa.

NSL-KDD zawiera prawie 5000000 próbek, co czyni go odpowiednim na potrzeby uczenia maszynowego, ale nie tak przytłaczająco wielkim, że zmusza badaczy do dzielenia go na mniejsze części, co zazwyczaj odbywa się w sposób losowy. Dzięki tej właściwości wyniki badań są bardziej porównywalne.

NSL-KDD został oczyszczony z powtarzających się próbek, co było problemem w KDD'99, mogąc powodować błąd stronniczości (ang. *bias*) wielu algorytmów ML.

Inne problemy w KDD'99 które rozwiązano w NSL-KDD:

– usunięto zbędne próbki;

Tablica 2. Nazwy klas i typy ataków w zbiorze NSL-KDD

Kategoria Ataków	Nazwa Klasy
Probe	ipsweep, mscan, nmap, portsweep, saint, satan
Denial of Service	apache, back, land, mailbomb, neptune, pod, processtable, smurf, teardrop, udpstorm
User-to-root (U2R)	buffer_overflow, loadmodule, perl, rootkit, ps, sqlattack, xterm
Remote-to-Local (R2L)	ftp_write, guess_passwd, imap, multihop

– wyeliminowano powtarzające się próbki;
– ulepszono proporcje liczby próbek w różnych klasach o różnym stopniu trudności;
– liczba próbek jest odpowiednia dla ML, dzięki czemu badacze nie muszą losowo dobierać próbek ze zbioru, dzięki czemu wyniki różnych badań są lepiej porównywalne.

Pomimo, że NSL-KDD nie jest zapisem rzeczywistego ruchu sieciowego, przez tyle lat cieszył się tak dużym powodzeniem wśród badaczy, że jest teraz głównym zbiorem przeznaczonym do porównywania i spójnej ewaluacji systemów wykrywania ataków sieciowych [111].

4.1.2. Zbiór ewaluacji systemów wykrywania ataków sieciowych - CIC IDS 2017

Zbiór CICIDS2017 [102] stworzono jako odpowiedź na brak godnych zaufania i adekwatnie nowych zbiorów z dziedziny cyberbezpieczeństwa. Zbiory IDS są słynne z bycia ciężkimi do zdobycia, a te dostępne naukowcom zazwyczaj wykazują zestaw swoich własnych problemów, takich jak brak różnorodności ruchu sieciowego, niewystarczającej rozmaitości ataków, niedostatecznej liczbie cech, i podobne.

Autorzy CICIDS2017 oferują zbiór zawierający realistyczny ruch podkładkowy (ang. "*background traffic*"), stworzony przez abstrakcję zachowania 25 użytkowników w wybranym zakresie protokołów. Dane zbierane były na przestrzeni 5 dni, z których przez 4 dni testowano zakres ataków,

Tablica 3. Wykorzystane kody etykiet i liczby próbek w odpowiednich klasach w CICIDS2017

Liczba próbek	Etykieta klasy	Zakodowana etykieta
1459377	BENIGN	0
207112	DoS Hulk	4
142924	PortScan	9
115222	DDoS	2
9264	DoS GoldenEye	3
7141	FTP-Patator	7
5307	SSH-Patator	10
5216	DoS slowloris	6
4949	DoS Slowhttptest	5
2713	Web Attack Brute Force	11
1760	Bot	1
1174	Web Attack XSS	13
38	Web Attack SQL Injection	12
10	Heartbleed	8

w tym cały asortyment złośliwego oprogramowania (ang. *malware*), ataków DoS, ataków w warstwie aplikacji i innych.

CICIDS2017 jest w chwili obecnej jednym z najnowszych zbiorów IDS dostępnych naukowcom, do tego zawiera ponad 80 cech z kategorii *NetFlow*. Wskaźnik zbalansowania klasy większościowej do sumy liczb wszystkich pozostałych klas wynosi 2,902, co świadczy o braku równowagi wśród klas w tym zbiorze)

Konkretne liczby próbek dla poszczególnych klas w zbiorze przeznaczonym do uczenia ukazane są w tab. 3.

4.2. Wykorzystane technologie: TensorFlow i Keras

W niniejszej pracy wykorzystano **TensorFLow** [2], otwartą bibliotekę, którą ze społecznością badaczy sztucznej inteligencji dzielą się naukowcy i inżynierowie pracujący dla zespołu *Google Brain*. Biblioteka ta dostarcza potężnych narzędzi z dziedziny uczenia maszynowego i uczenia głębokiego, oferując jednocześnie wyśmienite osiągi. TensorFlow jest w tej chwili zaimplementowany w szeregu naukowych i przemysłowych zastosowań.

Keras [26] jest łatwą w użyciu, darmową biblioteką będącą znakomitym interfejsem do TensorFlow i całego zastępu innych bibliotek. Keras pozwala użytkownikowi na niesamowite tempo prototypowania poprzez swoją modularność. Keras ma teraz pełne wsparcie ze strony TensorFlow i pozwala

na intuicyjne programowanie tak algorytmów uczenia maszynowego, jak i procedur głębokiego uczenia.

Eksperymenty wykonano na komputerze wyposażonym w procesor 7 generacji Intel Core i7-7500U z dwoma rdzeniami o taktowaniu 2.7 i 2.9GHz, oraz 16GB RAM. Użyto TensorFlow i Keras działające w środowisku Python 3.5.

4.3. Wartości ewaluacyjne

Wynik zadania klasyfikacyjnego może mieć jedną z czterech możliwych form:

- Wynik prawdziwie dodatni (ang. *True Positive* - TP) - liczba poprawnych klasyfikacji klasy szkodliwej
- Wynik prawdziwie ujemny (ang. *True Negative* - TN) - liczba poprawnych klasyfikacji klasy nieszkodliwej
- Wynik fałszywie dodatni (ang. *False Positive* - FP) - liczba próbek nieszkodliwych błędnie zaklasyfikowanych jako ataki
- Wynik fałszywie dodatni (ang. *False Negative* FN) - liczba próbek szkodliwych błędnie zaklasyfikowanych jako nieszkodliwe

Z powyższych wartości można uzyskać szereg pomocnych parametrów, jak *accuracy* (skuteczność), *precision* (precyzja), *recall* (rewokacja) i *f1-score*.

$$Accuracy(ACC) = \frac{TP + TN}{TP + TN + FN + FP} \tag{5}$$

$$Precision = \frac{TP}{TN + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1_{Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{8}$$

Warto zwrócić uwagę, że *precision* i *recall* w pewnym sensie stanowią parę, gdzie *precision* odpowiada za uwidocznienie błędów typu *false positive* (lub ich braku), a wysoki *recall* tłumaczy się na mniej błędów typu *false negative*. Stąd wynika, że można by wysterować model tak, by zwiększyć *recall* klasyfikatora, przez co byłby bardziej stanowczy w zatrzymywaniu potencjalnych ataków, powodując tym samym spadek wartości *precision*, ponieważ wzrośnie liczba błędów typu *false positive*.

Wartość *f1*, znana też jako *f1-score*, lub *miara F*, łączy w sobie wyniki wartości *precision* i *recall*. W uproszczeniu, im wyższe *f1*, tym lepsze osiągi klasyfikatora.

4.4. Walidacja krzyżowa - *Cross-Validation*

Uczenie maszynowe i różne odmiany sztucznej inteligencji doświadczają pewnych problemów. Jednym z nich jest przetrenowanie (ang. *overfitting*).

Modele zmagające się z tym problemem mogą osiągać znakomite rezultaty w laboratorium, ale zupełnie nie spełniać oczekiwań po wdrożeniu, gdy działają na nowych danych. W celu mitygacji tego fenomenu model poddaje się procedurze zwanej sprawdzaniem krzyżowym (ang. *cross-validation*).

W ramach takiej procedury zbiór danych dzieli się na k części, z których *k-1* wykorzystywane jest na uczenie, a pozostała na weryfikację modelu. Następnie procedura powtarzana jest k razy, zmieniając część służącą do weryfikacji. Części te są dobierane ze zbioru losowo [52][47].

4.5. Wpływ metod równoważenia zbiorów na modele oparte o uczenie maszynowe - wyniki

Zbiór CICIDS2017 składa się z 13 klas, 12 z nich to ataki, 1 jest klasą nieszkodliwego ruchu (etykieta "BENIGN"). Jak zilustrowano na rys. 9, istnieje znaczna rozbieżność pomiędzy liczbami próbek w zbiorze, zwłaszcza klasa "BENIGN" jest znacznie większa w porównaniu do klas zawierających ataki. Liczba próbek w konkretnych klasach w tym zbiorze została ukazana w tab. 3. Dane zebrane w tab. 3 przedstawiają wyniki *precision*, *recall* i *f1-score* osiągane przez każdy ze wspomnianych klasyfikatorów - ANN, RF i *NaiveBayes* dla wszystkich 13 klas zawartych w zbiorze CICIDS2017, kolumna *support* zawiera liczby próbek pochodzących z konkretnych klas w zbiorze testowym. Wartość *macro avg* oznacza średnią arytmetyczną wartości *precision*, *recall* i *f1-score* wszystkich klas, *weighted avg* jest średnią ważoną uwzględniającą liczbę próbek w konkretnych klasach (z kolumny *support*).

W trakcie eksperymentów wstępna hipoteza zakładała, że metody równoważenia liczby próbek w klasach poprawią ogólne wyniki klasyfikatorów. W celu obserwacji wpływu jaki balansowanie danych ma na wyniki trzech referencyjnych algorytmów ML - sztucznej sieci neuronowej, algorytmu *Random Forest* i klasyfikatora *Naïve Bayes*, wykorzystano technikę *Random Subsampling* (tab. 5), oraz szereg innych metod z kategorii "*undersampling*". Dla pełniejszego obrazu użyto również metody z kategorii

"*oversampling*", jest nią *Borderline SMOTE*. Wyniki działania tych metod ukazano w tab. 6, 9, 7 i 8.

Z wyników "*recall*" niezbalansowanego zbioru (tab. 3) wyraźnie wynika, że pewne próbki klas mniejszościowych nie są rozpoznawane w odpowiedni sposób (np. klasa 11 i 13). Procedura równoważenia klasy "BENIGN" tak, by posiadała taką samą liczbę próbek jak suma wszystkich liczb próbek klas zawierających ataki zmieniła tak wartość "*precision*" jak i "*recall*" osiągane przez algorytm.

Po szeregu eksperymentów okazało się, że żadna z bardziej złożonych metod z kategorii "*undersampling*" nie osiąga wyników lepszych od tych pochodzących z wykorzystania techniki "*random subsampling*", w wypadku zbioru CICIDS2017. Testy uwidoczniły też interesujące połączenie pomiędzy wartościami *precision*, *recall* i wskaźnikiem braku balansu zbioru.

Wykryto związek przyczynowo-skutkowy pomiędzy "*precision*" i "*recall*" w odpowiednich klasach a kontrolującą je liczbą próbek odpowiednich klas w zbiorze. W celu zbadania takiego twierdzenia przeprowadzono serię testów. Algorytm *Random Forest* został nauczony przy pomocy niezbalansowanego zbioru CICIDS2017, następnie kolejno na wszystkich klasach przeprowadzano "*random subsampling*" tak, by liczba instancji w klasach większościowych wynosiła tyle, co w największej mniejszościowej, dobierając coraz mniejsze klasy (tab. 11 - 1174 próbek w klasach większościowych, tab. 10 - 7141 próbek w klasach większościowych)

Eksperymenty potwierdziły, że zmienianie stosunku liczby próbek między klasami poprzez "*undersampling*" klas większościowych poprawia "*recall*" wykrywania klas mniejszościowych, ale pogarsza "*precision*" próbkowanych klas większościowych. Oznacza to, że równoważenie zbioru powoduje, że algorytmy ML przeklasyfikowują całe porcje próbek z klas większościowych do klas mniejszościowych, zwiększając zarówno liczbę dobrze sklasyfikowanych instancji klas mniejszościowych, jak i błędnie sklasyfikowanych instancji klas większościowych, czyli powodując błędy typu "*false positive*".

W następnym eksperymencie do algorytmu *Random Forest* dodano komponent pozwalający ustanowić wagę dla poszczególnych klas. Taki Random Forest z rodzaju "*cost-sensitive*" osiągnął wyniki lepsze od którejkolwiek z metod z grupy działającej na danych (tab. 12). Zwłaszcza warto zwrócić uwagę, ze osiągnięta wartość "*recall*" dla klasy 13 jest wyższa, jednocześnie zachowując względnie wysoką wartość "*precision*" - w porównaniu do pozostałych przebadanych metod. Wykryto również związek pomiędzy klasami 11 i 13, w którym ustawienie wyższej wagi dla klasy 13 powodowało błędną klasyfikację próbek z klasy 11 jako próbek klasy 13 i odwrotnie. Te klasy to odpowiednio Web Attack-Brute Force i Web

Tablica 4. CICIDS2017 (pełen zbiór) / Niezbalansowany

	ANN ACC: 0.9833			RandomForest ACC: 0.9987			NaïveBayes ACC: 0.2905			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	0.99	0.99	0.99	1.00	1.00	1.00	1.00	0.10	0.18	162154
1	0.97	0.35	0.52	0.88	**0.68**	0.77	0.01	0.65	0.01	196
2	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.95	0.94	12803
3	0.99	0.97	0.98	1.00	0.99	1.00	0.09	0.93	0.16	1029
4	0.95	0.94	0.94	1.00	1.00	1.00	0.74	0.70	0.72	23012
5	0.89	0.98	0.93	0.96	0.98	0.97	0.00	0.67	0.01	550
6	0.99	0.98	0.99	1.00	0.99	0.99	0.05	0.52	0.09	580
7	0.99	0.98	0.99	1.00	1.00	1.00	0.10	0.99	0.18	794
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1
9	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	15880
10	1.00	0.49	0.66	1.00	1.00	1.00	0.08	0.99	0.15	590
11	0.85	0.10	0.17	0.86	**0.99**	0.92	0.00	0.07	0.00	301
12	0.00	0.00	0.00	1.00	1.00	1.00	0.01	1.00	0.02	4
13	1.00	0.02	0.05	0.95	**0.61**	0.74	0.08	0.93	0.14	130
macro avg	0.90	0.70	0.73	0.97	0.95	0.96	0.36	0.75	0.33	218024
weighted avg	0.98	0.98	0.98	1.00	1.00	1.00	0.95	0.29	0.34	218024

Tablica 5. CICIDS2017 / Random Subsampling

			ANN ACC: 0.9812			RandomForest ACC: **0.9980**			NaïveBayes ACC: 0.2911	
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	1.00	0.98	0.99	1.00	1.00	1.00	1.00	0.10	0.18	162154
1	0.50	0.63	0.56	0.91	**0.92**	0.91	0.01	0.65	0.01	196
2	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.95	0.94	12803
3	0.98	0.98	0.98	1.00	1.00	1.00	0.09	0.93	0.16	1029
4	0.90	0.99	0.95	1.00	1.00	1.00	0.74	0.70	0.72	23012
5	0.90	0.99	0.94	0.98	0.99	0.99	0.00	0.67	0.01	550
6	0.97	0.98	0.97	0.99	0.99	0.99	0.05	0.52	0.09	580
7	0.99	0.98	0.98	1.00	1.00	1.00	0.10	0.99	0.19	794
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1
9	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	15880
10	0.97	0.49	0.65	1.00	0.99	1.00	0.08	0.99	0.15	590
11	0.59	0.23	0.33	0.80	**0.97**	0.88	0.00	0.07	0.00	301
12	0.00	0.00	0.00	1.00	0.80	0.89	0.01	1.00	0.02	4
13	0.80	0.03	0.06	0.96	**0.40**	0.57	0.08	0.93	0.15	130
macro avg	0.83	0.73	0.74	0.97	0.93	0.94	0.36	0.75	0.33	218024
weighted avg	0.99	0.98	0.98	1.00	1.00	1.00	0.95	0.29	0.34	218024

Tablica 6. CICIDS2017 / NearMiss

	ANN ACC: **0.7725**			RandomForest ACC: 0.7116			Naïve Bayes ACC: 0.3744			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	1.00	0.70	0.82	1.00	0.61	0.76	1.00	0.21	0.35	162154
1	0.02	0.71	0.04	0.03	0.81	0.06	0.01	1.00	0.02	196
2	0.90	1.00	0.95	0.52	1.00	0.68	0.91	0.96	0.93	12803
3	0.99	0.97	0.98	0.97	0.99	0.98	0.22	0.93	0.35	1029
4	0.66	1.00	0.80	0.51	1.00	0.68	0.65	0.70	0.68	23012
5	0.58	0.99	0.73	0.57	0.98	0.72	0.00	0.64	0.01	550
6	0.27	0.98	0.43	0.07	0.99	0.13	0.07	0.82	0.13	580
7	0.19	1.00	0.32	0.25	1.00	0.40	0.10	1.00	0.18	794
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1
9	0.45	1.00	0.62	0.89	1.00	0.94	1.00	0.99	0.99	15880
10	0.12	0.99	0.21	0.07	1.00	0.13	0.11	0.99	0.20	590
11	0.35	**0.56**	0.43	0.09	0.99	0.16	0.00	0.08	0.01	301
12	0.00	0.00	0.00	0.05	1.00	0.10	0.01	1.00	0.02	4
13	0.01	**0.02**	0.02	0.06	0.49	0.11	0.17	0.92	0.29	130
macro avg	0.47	0.78	0.52	0.43	0.92	0.49	0.38	0.80	0.37	218024
weighted avg	0.91	0.77	0.81	0.90	0.71	0.75	0.94	0.37	0.46	218024

Tablica 7. CICIDS2017 / Tomek-Links

	ANN ACC: 0.9836			RandomForest ACC: **0.9986**			Naïve Bayes ACC: 0.5263			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	0.99	0.99	0.99	1.00	1.00	1.00	1.00	0.10	0.18	162154
1	0.92	0.37	0.53	0.81	0.78	0.80	0.01	0.65	0.01	196
2	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.95	0.94	12803
3	0.99	0.97	0.98	1.00	0.99	0.99	0.09	0.93	0.16	1029
4	0.94	0.95	0.95	1.00	1.00	1.00	0.74	0.70	0.72	23012
5	0.90	0.99	0.94	0.97	0.98	0.98	0.00	0.67	0.01	550
6	0.99	0.98	0.98	0.99	0.99	0.99	0.05	0.52	0.09	580
7	0.99	0.98	0.99	1.00	1.00	1.00	0.10	0.99	0.18	794
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1
9	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	15880
10	1.00	0.49	0.66	1.00	0.99	1.00	0.08	0.99	0.15	590
11	0.85	0.07	0.13	0.84	**0.97**	0.90	0.00	0.07	0.00	301
12	0.00	0.00	0.00	1.00	0.75	0.86	0.01	1.00	0.02	4
13	1.00	0.02	0.05	0.91	**0.55**	0.68	0.08	0.93	0.14	130
macro avg	0.90	0.70	0.73	0.97	0.93	0.94	0.36	0.75	0.33	218024
weighted avg	0.98	0.98	0.98	1.00	1.00	1.00	0.95	0.29	0.34	218024

Tablica 8. CICIDS2017 / ClusterCentroids

			ANN			RandomForest			NaïveBayes	
			ACC: **0.4569**			ACC: 0.2560			ACC: 0.2832	
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	1.00	0.47	0.64	1.00	0.00	0.00	1.00	0.09	0.16	162154
1	0.01	0.28	0.02	0.03	1.00	0.07	0.01	0.65	0.01	196
2	0.90	0.63	0.74	0.74	1.00	0.85	0.93	0.95	0.94	12803
3	0.77	0.68	0.72	0.75	1.00	0.85	0.08	0.93	0.16	1029
4	0.86	0.62	0.72	0.81	1.00	0.90	0.67	0.70	0.69	23012
5	0.15	0.69	0.25	0.82	0.99	0.89	0.00	0.67	0.01	550
6	0.35	0.22	0.27	0.25	0.99	0.40	0.05	0.52	0.09	580
7	0.06	0.47	0.11	0.71	1.00	0.83	0.10	0.99	0.18	794
8	0.01	1.00	0.01	0.50	1.00	0.67	1.00	1.00	1.00	1
9	0.57	0.00	0.00	1.00	1.00	1.00	1.00	0.99	0.99	15880
10	0.00	0.00	0.00	0.10	1.00	0.18	0.08	0.99	0.15	590
11	0.00	0.00	0.00	0.17	0.98	0.29	0.00	0.07	0.00	301
12	0.00	0.00	0.00	0.18	1.00	0.31	0.01	1.00	0.02	4
13	0.00	0.03	0.00	0.05	0.65	0.09	0.09	0.93	0.16	130
macro avg	0.34	0.36	0.25	0.51	0.90	0.52	0.36	0.75	0.33	218024
weighted avg	0.93	0.46	0.60	0.95	0.26	0.23	0.94	0.28	0.32	218024

Tablica 9. CICIDS2017 / BORDERLINE SMOTE

	ANN ACC: 0.9753			RandomForest ACC: **0.9920**			NaïveBayes ACC: 0.5263			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
0	1.00	0.97	0.99	1.00	0.99	1.00	1.00	0.52	0.68	162154
1	0.17	0.94	0.29	0.17	0.98	0.30	0.00	0.65	0.01	196
2	0.99	1.00	1.00	1.00	1.00	1.00	0.85	0.95	0.90	12803
3	0.94	0.99	0.96	1.00	1.00	1.00	0.05	0.87	0.10	1029
4	0.93	0.99	0.96	0.99	1.00	1.00	0.68	0.70	0.69	23012
5	0.64	0.96	0.77	0.94	0.98	0.96	0.01	0.20	0.02	550
6	0.78	0.51	0.62	1.00	0.97	0.98	0.01	0.03	0.01	580
7	0.87	0.98	0.92	0.99	1.00	1.00	0.02	0.47	0.05	794
8	0.50	1.00	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1
9	0.99	1.00	0.99	1.00	1.00	1.00	0.01	0.00	0.00	15880
10	0.64	0.53	0.58	1.00	0.89	0.94	0.07	0.50	0.12	590
11	0.20	0.26	0.22	0.85	**0.84**	0.84	0.02	0.89	0.05	301
12	0.01	0.75	0.01	1.00	1.00	1.00	0.01	1.00	0.03	4
13	0.16	0.77	0.27	0.67	**0.90**	0.77	0.00	0.00	0.00	130
macro avg	0.63	0.83	0.66	0.90	0.97	0.91	0.27	0.56	0.26	218024
weighted avg	0.98	0.98	0.98	1.00	0.99	0.99	0.87	0.53	0.64	218024

Tablica 10. CICIDS2017 / Random Subsampling do 7141 próbek w klasie / RandomForest

	precision	recall	f1-score	support
0	1.00	0.98	0.99	162154
1	0.13	0.99	0.23	196
2	1.00	1.00	1.00	12803
3	0.92	1.00	0.96	1029
4	0.98	1.00	0.99	23012
5	0.85	0.99	0.92	550
6	0.93	0.99	0.96	580
7	0.93	1.00	0.96	794
8	0.17	1.00	0.29	1
9	1.00	1.00	1.00	15880
10	0.73	1.00	0.85	590
11	0.63	**0.98**	0.77	301
12	0.07	1.00	0.14	4
13	0.32	**0.48**	0.39	130
accuracy			**0.9872**	218024
macro avg	0.69	0.96	0.74	218024
weighted avg	0.99	0.99	0.99	218024

Tablica 11. CICIDS2017 / Random Subsampling do 1174 próbek w klasie / RandomForest

	precision	recall	f1-score	support
0	1.00	0.96	0.98	162154
1	0.07	1.00	0.13	196
2	0.99	1.00	1.00	12803
3	0.69	1.00	0.82	1029
4	0.94	0.99	0.97	23012
5	0.76	0.99	0.86	550
6	0.86	0.99	0.92	580
7	0.81	1.00	0.89	794
8	0.17	1.00	0.29	1
9	1.00	1.00	1.00	15880
10	0.44	1.00	0.61	590
11	0.23	**0.65**	0.34	301
12	0.07	1.00	0.13	4
13	0.13	**0.95**	0.23	130
accuracy			**0.9657**	218024
macro avg	0.58	0.97	0.65	218024
weighted avg	0.99	0.97	0.97	218024

Tablica 12. CICIDS2017 / Cost-Sensitive RandomForest

	precision	recall	f1-score	support
0	1.00	1.00	1.00	162154
1	0.34	0.91	0.50	196
2	1.00	1.00	1.00	12803
3	1.00	0.99	0.99	1029
4	1.00	1.00	1.00	23012
5	0.97	0.98	0.97	550
6	1.00	0.99	0.99	580
7	1.00	1.00	1.00	794
8	1.00	1.00	1.00	1
9	1.00	1.00	1.00	15880
10	1.00	1.00	1.00	590
11	0.98	**0.85**	0.91	301
12	1.00	1.00	1.00	4
13	0.72	**0.96**	0.83	130
accuracy			**0.9973**	218024
macro avg	0.93	0.98	0.94	218024
weighted avg	1.00	1.00	1.00	218024

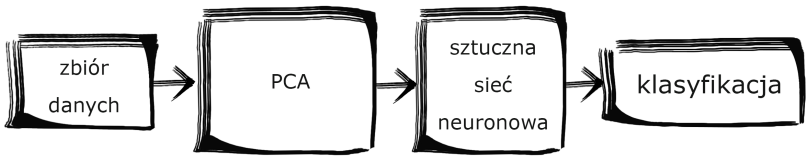
Attack-XSS. Może to świadczyć o fakcie, tych dwóch grup ataków nie da się wyraźnie rozróżnić na podstawie cech zawartych w CICIDS2017.

4.6. Równoważenie danych - podsumowanie

W niniejszym dziale pracy przeprowadzono i zaprezentowano ewaluację szeregu sposobów rozwiązywania problemu niezbalansowanego zbioru danych w kontekście wykrywania ataków sieciowych. Przeprowadzone eksperymenty pozwoliły na zauważenie wielu interesujących szczegółów dotyczących wpływu tych metod na metody oparte na algorytmach uczenia maszynowego. W pierwszej kolejności, w wypadku zbioru ataków sieciowych CICIDS2017, metoda "*random subsampling*" okazała się być równie skuteczna lub lepsza co inne, bardziej wyszukane metody z grupy "*undersampling*". Jej wyniki są porównywalne do *Borderline SMOTE*.

Po drugie, ostatecznie proporcje zbioru wykorzystanego do uczenia mogą mieć równie duży wpływ na wyniki osiągane przez metody oparte o algorytmy ML, co sam wybór procedury balansującej dane.

Po trzecie, między wielkością klasy większościowej a wartością "*precision*" i "*recall*" osiąganymi przez klasyfikator istnieje relacja, która znajduje wyraz w liczbie próbek klasy większościowej błędnie zaklasyfikowanych jako



Rysunek 16. Proces klasyfikacji dla wybranego zbioru. Podobny proces zastosowano dla zbioru NSL-KDD i CICIDS2017

klasa mniejszościowa (lub próbek klasy mniejszościowej błędnie zaklasyfikowanych jako większościowa).

4.7. Propozycja optymalizacji hiperparametrów sztucznych sieci neuronowych

w trakcie eksperymentów przetestowano szereg możliwych ustawień architektury, zaczynając od jednej ukrytej warstwy o 25 neuronach (liczba wyznaczona przez podzielenie liczby cech podawanych na warstwę wejściową sieci na pół), przez architektury o dwóch, trzech, czterech itd. aż do 7 warstw z różnymi liczbami neuronów, włącznie z liczbą neuronów wyższą od liczby wejść. Przykład wyników dla architektury o 4 warstwach po 25 neuronów znajduje się w tab. 20.

Zbiór NSL-KDD oferuje 41 cech, z czego 3 to cechy jakościowe (categorical variables). Ponieważ częścią opracowanej metody uczenia maszynowego jest PCA, które opiera się na analizie wariancji, zdecydowano wykluczyć te cechy ze zbioru. Podobny proces eliminacji wykonano na CICIDS2017.

W następnym kroku wykonywano PCA w celu ograniczenia liczby cech do najważniejszych komponentów podawanych na warstwę wejściową ANN. Proces ten zilustrowano na rys.16

W pierwszym eksperymencie na podzbiorze "Wtorek" ze zbioru CICIDS2017 wstępne wyniki (tab.17) sprawiały wrażenie bardzo obiecujących, pomimo braku równowagi między klasami. Wartość "*recall*" dla jednej z klas na poziomie 0.54 sugerowała jednak, że duża część próbek tej klasy jest klasyfikowana jako próbki klasy większościowej.

Różne techniki przeciwdziałania problemowi braku równowagi między klasami w CICIDS2017 omówiono we wcześniejszym dziale. Wynik dla podzbioru "Wtorek" zrównoważonego przez random subsampling klasy "BENIGN" dla łatwiejszego porównania z przykładem powyższym umieszczono w tab. 18.

Tablica 13. Wyniki optymalizacji - 1 ukryta warstwa, 25 neuronów, NSL-KDD

Accuracy	Activation	Batch Size	Epochs	Optimizer
0.9986	relu	10	300	rmsprop
0.9989	relu	10	300	adam
0.9983	relu	10	300	SGD
0.9989	relu	100	300	rmsprop
0.9989	relu	100	300	adam
0.9957	relu	100	300	SGD
0.9987	sigmoid	10	300	rmsprop
0.9990	sigmoid	10	300	adam
0.9961	sigmoid	10	300	SGD
0.9989	sigmoid	100	300	rmsprop
0.9991	sigmoid	100	300	adam
0.9921	sigmoid	100	300	SGD
0.9986	tanh	10	300	rmsprop
0.9990	tanh	10	300	adam
0.9980	tanh	10	300	SGD
0.9990	tanh	100	300	rmsprop
0.9991	tanh	100	300	adam
0.9953	tanh	100	300	SGD
0.9987	hard_sigmoid	10	300	rmsprop
0.9990	hard_sigmoid	10	300	adam
0.9961	hard_sigmoid	10	300	SGD
0.9989	hard_sigmoid	100	300	rmsprop
0.9991	hard_sigmoid	100	300	adam
0.9919	hard_sigmoid	100	300	SGD

Tablica 14. Wyniki optymalizacji, 2 ukryte warstwy, 25 neuronów każda, zbiór NSL-KDD

Accuracy	Optimizer	Batch Size	Epochs	Activation
0.9991	adam	10	300	relu
0.9988	rmsprop	10	300	relu
0.9991	adam	10	300	relu
0.9988	rmsprop	100	300	relu
0.9990	adam	100	300	relu
0.9957	SGD	100	300	relu
0.9987	rmsprop	10	300	sigmoid
0.9990	adam	10	300	sigmoid
0.9954	SGD	10	300	sigmoid
0.9991	rmsprop	100	300	sigmoid
0.9991	adam	100	300	sigmoid
0.9901	SGD	100	300	sigmoid
0.9988	rmsprop	10	300	tanh
0.9989	adam	10	300	tanh
0.9986	SGD	10	300	tanh
0.9990	rmsprop	100	300	tanh
0.9990	adam	100	300	tanh
0.9954	SGD	100	300	tanh
0.9988	rmsprop	10	300	hard_sigmoid
0.9990	adam	10	300	hard_sigmoid
0.9953	SGD	10	300	hard_sigmoid
0.9990	rmsprop	100	300	hard_sigmoid
0.9990	adam	100	300	hard_sigmoid
0.9899	SGD	100	300	hard_sigmoid

Tablica 15. Wyniki optymalizacji - 1 ukryta warstwa, 10 neuronów, zbiór NSL-KDD

Accuracy	Optimizer	Epochs	Activation	Batch Size
0.9989	adam	300	sigmoid	10
0.9985	rmsprop	300	relu	10
0.9988	adam	300	relu	10
0.9974	SGD	300	relu	10
0.9988	rmsprop	300	relu	100
0.9989	adam	300	relu	100
0.9952	SGD	300	relu	100
0.9984	rmsprop	300	sigmoid	10
0.9989	adam	300	sigmoid	10
0.9957	SGD	300	sigmoid	10
0.9987	rmsprop	300	sigmoid	100
0.9987	adam	300	sigmoid	100
0.9910	SGD	300	sigmoid	100
0.9986	rmsprop	300	tanh	10
0.9988	adam	300	tanh	10
0.9968	SGD	300	tanh	10
0.9988	rmsprop	300	tanh	100
0.9988	adam	300	tanh	100
0.9941	SGD	300	tanh	100
0.9985	rmsprop	300	hard_sigmoid	10
0.9988	adam	300	hard_sigmoid	10
0.9956	SGD	300	hard_sigmoid	10
0.9987	rmsprop	300	hard_sigmoid	100
0.9988	adam	300	hard_sigmoid	100
0.9914	SGD	300	hard_sigmoid	100

4.8. Optymalizacja hiperparametrów - podsumowanie

Optymalizacja hiperparametrów została wykonana na każdej z testowanych architektur. Metoda grid search sprawdza wszystkie możliwe kombinacje wybranych hiperparametrów. W eksperymentach wykorzystano różne liczby epoch, batch size, różne optymizatory i różne funkcje aktywacji, w różnych kombinacjach w celu uzyskania najwyższej możliwej skuteczności. Załączone tabele uwidaczniają sposób w jaki sposób skuteczność zmienia się przy różnych ustawieniach ANN na tej samej architekturze. Przykładowo tab. 20 ilustruje wyniki procedury grid search przeprowadzonej na ANN o 4 ukrytych warstwach, po 25 neuronów na każdej.

tab. 13 pokazuje wyniki procedury grid search dla sieci ANN z jedną ukrytą warstwą o 25 neuronach, pozostałe tab. 14, 15 pokazują szczegóły dla sieci ANN z dwiema ukrytymi warstwami po 25 neuronów i o jednej ukrytej warstwie z 10 neuronami.

Optymalne ustawienie algorytmu dla zbioru CICIDS2017 również zostało określone przy pomocy procedury grid search. Przykładowe wyniki tego procesu można odnaleźć na tab.16. Tak, jak w wypadku optymalizacji hiperparametrów dla zbioru NSL-KDD, wyniki dla poszczególnych ustawień różnią się od siebie znacząco.

W tym dziale w celach porównawczych mieszczono wyniki innych podejść opartych o uczenie maszynowe. Testy wykonano z wykorzystaniem algorytmów *Support Vector Machine* (SVM), klasyfikatorów *Naïve Bayes* i *Adaptive Boosting* (ADABoost). W tab. 19 znaleźć można różnice pomiędzy wynikami tych klasyfikatorów na zbiorze CICIDS2017. Jak można zauważyć w trakcie inspekcji tej tabeli, przedstawione metody oparte o sztuczne sieci neuronowe, jak i algorytm random forest osiągają podobne wyniki w wartości *recall*, wyprzedzają jednak inne metody uczenia maszynowego w *precision* i *accuracy*.

Istotnym wkładem w dziedzinę jest wykorzystanie optymizacji hiperparametrów w sztucznych sieciach neuronowych dla wykrywania ataków sieciowych w nowoczesnych danych, a wagę i efektywność procesu zademonstrowano na dwóch wzorcowych zbiorach danych.

W niniejszej pracy zaprezentowano tę metodę ze szczegółami, demonstrując jej wydajność szeregiem wyników. Rozwiązano też problem braku balansu w danych.

Jak wykazano na załączonych tablicach, ostateczny dobór hiperparametrów może mieć krytyczny efekt na osiągane przez algorytm ANN wyniki. Dla przykładu, najlepsza uzyskana skuteczność dla zbioru NSL-KDD wynosi 0.999090 przy korzystaniu z optymizatora *ADAM*, *batch size* 10, 300 epok i *Rectified Linear Unit* jako funkcja aktywacji, na architekturze

Tablica 16. 4 ukryte warstwy, 25 neuronów każda, zbiór CICIDS2017

Accuracy	Activation	Batch Size	Optimizer	Epochs
0.9752	relu	50	adam	30
0.9770	relu	50	rmsprop	30
0.9706	relu	50	SGD	30
0.9743	relu	100	adam	30
0.9767	relu	100	rmsprop	30
0.9582	relu	100	SGD	30
0.9705	sigmoid	50	adam	30
0.9698	sigmoid	50	rmsprop	30
0.9302	sigmoid	50	SGD	30
0.9691	sigmoid	100	adam	30
0.9656	sigmoid	100	rmsprop	30
0.5850	sigmoid	100	SGD	30
0.9724	hard_sigmoid	50	adam	30
0.9734	hard_sigmoid	50	rmsprop	30
0.7085	hard_sigmoid	50	SGD	30
0.9719	hard_sigmoid	100	adam	30
0.9726	hard_sigmoid	100	rmsprop	30
0.4998	hard_sigmoid	100	SGD	30

Tablica 17. Niezbalansowany CICIDS2017, podzbiór “Wtorek”, ANN

	precision	recall	f1-score	support
0	0.99	1.00	1.00	43171
1	1.00	0.98	0.99	820
2	1.00	**0.54**	0.70	574
micro avg	0.99	0.99	0.99	44565
macro avg	1.00	0.84	0.89	44565
weighted avg	0.99	0.99	0.99	44565
samples avg	0.99	0.99	0.99	44565

Tablica 18. Zbalansowany CICIDS2017 podzbiór “Wtorek”

	precision	recall	f1-score	support
0	0.99	0.96	0.98	1387
1	1.00	1.00	1.00	804
2	0.92	**0.99**	0.95	576
micro avg	0.98	0.98	0.98	2767
macro avg	0.97	0.98	0.98	2767
weighted avg	0.98	0.98	0.98	2767

Tablica 19. Wstępne wyniki innych metod ML, CICIDS2017, podzbiór "Wtorek", zbiór zbalansowany metodą random subsampling

	precision	recall	f1-score	support
SVM , Accuracy: 0.9584				
0	1.00	0.92	0.96	1387
1	0.97	1.00	0.98	804
2	0.87	0.99	0.93	576
micro avg	0.96	0.96	0.96	2767
macro avg	0.95	0.97	0.96	2767
weighted avg	0.96	0.96	0.96	2767
Naïve Bayes , Accuracy: 0.9078				
0	1.00	0.82	0.90	1387
1	0.86	1.00	0.93	804
2	0.82	1.00	0.90	576
micro avg	0.91	0.91	0.91	2767
macro avg	0.89	0.94	0.91	2767
weighted avg	0.92	0.91	0.91	2767
ADABoost , Accuracy: 0.9382				
0	1.00	0.88	0.93	1387
1	1.00	1.00	1.00	804
2	0.78	0.99	0.87	576
micro avg	0.94	0.94	0.94	2767
macro avg	0.92	0.96	0.93	2767
weighted avg	0.95	0.94	0.94	2767

Tablica 20. Efekty optymalizacji - 4 ukryte warstwy, 25 neuronów na każdej, zbiór NSL-KDD

Accuracy	Epochs	Optimizer	Batch Size	Activation
0.9990	300	adam	10	sigmoid
0.9537	300	rmsprop	10	relu
0.9543	300	adam	10	relu
0.9498	300	SGD	10	relu
0.9545	300	rmsprop	100	relu
0.9551	300	adam	10	relu
0.8592	300	SGD	100	relu
0.9986	300	rmsprop	10	sigmoid
0.9970	300	adam	10	sigmoid
0.9922	300	SGD	10	sigmoid
0.9988	300	rmsprop	100	sigmoid
0.9990	300	adam	100	sigmoid
0.9576	300	SGD	100	sigmoid
0.2046	300	rmsprop	10	tanh
0.0564	300	adam	10	tanh
0.4200	300	SGD	10	tanh
0.1200	300	rmsprop	100	tanh
0.1227	300	adam	100	tanh
0.6200	300	SGD	100	tanh
0.9966	300	rmsprop	10	hard_sigmoid
0.9973	300	adam	10	hard_sigmoid
0.9625	300	SGD	10	hard_sigmoid
0.9985	300	rmsprop	100	hard_sigmoid
0.9985	300	adam	100	hard_sigmoid
0.9576	300	SGD	100	hard_sigmoid

stworzonej z 4 ukrytych warstw po 25 neuronów każda. Na tej samej architekturze, zaledwie po zmianie funkcji aktywacji na *hiperbolic tangent*, skuteczność osiąga wartość 0.0564. Oznacza to, że "prawie zoptymalizowany" model w tym przypadku wybierał złą odpowiedź praktycznie za każdym jednym razem.

Tablica 21. Wyniki sieci neuronowej GRU dla zbioru NSL-KDD

class name	precision	recall	f1-score	support
back	0.94	1.00	0.97	17
buffer_overflow	0.00	0.00	0.00	1
ftp_write	0.00	0.00	0.00	0
guess_passwd	0.00	0.00	0.00	3
imap	0.00	0.00	0.00	1
ipsweep	0.92	0.94	0.93	72
land	0.00	0.00	0.00	0
multihop	0.00	0.00	0.00	0
normal	1.00	1.00	1.00	808
nmap	0.92	0.59	0.72	37
neptune	0.98	0.99	0.99	1358
phf	0.00	0.00	0.00	0
pod	0.00	0.00	0.00	5
portsweep	0.93	0.93	0.93	55
rootkit	0.00	0.00	0.00	0
satan	0.99	0.92	0.95	72
smurf	0.98	0.95	0.96	55
spy	0.00	0.00	0.00	0
teardrop	1.00	1.00	1.00	21
warezclient	0.00	0.00	0.00	14
warezmaster	0.00	0.00	0.00	0
micro avg	0.98	0.97	0.98	2519
weighted avg	0.97	0.97	0.97	2519

4.9. Eksperymenty wykorzystujące architekturę Gated Recurrent Unit

W ramach eksperymentów użyto podziału zbioru danych w stosunku 9:1. Należy nadmienić, że przed podaniem danych do architektury GRU należy zmienić kształt danych na tablicę 3D.

W teście wykorzystano architekturę GRU o dwóch warstwach po 40 neuronów każda, z warstwami dropout ustawionymi na 0.2 zaraz po nich dla zbioru NSL-KDD, oraz podobną architekturę tylko z 78 neuronami na pierwszej ukrytej warstwie i 38 na drugiej dla CICIDS2017.

Tablica 22. Wyniki sieci neuronowej GRU dla zbioru CICIDS2017

	precision	recall	f1-score	support
0	0.99	0.98	0.98	55968
1	0.80	0.66	0.72	201
2	1.00	1.00	1.00	12834
3	0.93	0.97	0.95	1013
4	0.95	0.99	0.97	23013
5	0.94	0.94	0.94	582
6	0.94	0.98	0.96	561
7	0.99	0.99	0.99	788
8	0.00	0.00	0.00	0
9	1.00	1.00	1.00	15755
10	0.97	0.47	0.63	597
11	0.61	0.75	0.67	296
12	0.00	0.00	0.00	1
13	0.00	0.00	0.00	131
micro avg	0.98	0.98	0.98	111740
macro avg	0.72	0.69	0.70	111740
weighted avg	0.98	0.98	0.98	111740

4.10. Wyniki Gated Recurrent Unit

Wyniki uzyskane przez GRU na NSL-KDD są obiecujące, model oparty o tę architekturę osiągnął średnią ważoną 97% we wszystkich mierzonych wskaźnikach, dochodząc do 100% dla niektórych klas.

Niestety, nie wszystkie klasy udało się rozpoznać, jak to zilustrowano na tab. 21, niektóre z ataków wpadły w klasy o większej liczbie próbek.

Model oparty o architekturę GRU na zbiorze CICIDS2017 osiągnął wyniki, które zebrano w tab. 22. Jak łatwo zauważyć, osiągi na klasach z dużą liczbą wystąpień są w zakresie powyżej 0.9. Niestety pomimo podania do fazy uczenia zbilansowanej wersji CICIDS2017 dwie klasy o najmniejszej liczbie próbek zupełnie umknęły w procesie wykrywania, tak jak i połowa próbek z klasy 10 i 34% próbek z klasy 1. Niemniej jednak skuteczność na poziomie 0.9797 stawia go konkurencyjnym poziomie w stosunku do innych rozwiązań.

Unikatowa zdolność do wyławiania zależności czasowych w danych powoduje jednak, że nie należy tej architektury ostatecznie skreślać - w następnym dziale omówione zostaną propozycje wykorzystania GRU w sposób pozwalający na wykorzystanie zalet tej architektury, ale osiągnięcie wyższych wyników.

4.11. Podejście oparte o ekstrakcję cech przy pomocy architektury Gated Recurrent Unit i klasyfikacje przy pomocy Random Forest - eksperymenty i wyniki

W celu przeprowadzenia eksperymentu zbiór CICIDS2017 podzielono na 3 części. Pierwsza służy do nauki ekstraktora cech opartego o sieć GRU, druga część do sprawdzenia wyników sieci GRU i nauki algorytmu RF użytego w celu klasyfikacji. Trzecia do weryfikacji całego podejścia.

Sprawdzono dwa podejścia, ukazane na rys. 14 i rys. 15. Pierwsze, polegające na zbalansowaniu danych przedstawioną wcześniej metodą *Random Subsampling*, następnie wykorzystaniu sieci GRU w roli ekstraktora cech, pobierając aktywacje z przedostatniej warstwy jako cechy do dalszego przetwarzania. Po przejściu przez GRU dane zostają poddane PCA i ostatecznie trafiają do klasyfikatora, który zwraca wynik. Drugie podejście jest bardzo podobne, ale zamiast balansowania zbioru przed ekstrakcją cech wykorzystany został algorytm Random Forest z modyfikacją *cost-sensitive*.

Porównując wyniki w tab. 23, 24, 25, 26 i 27 łatwo zwrócić uwagę, że wszystkie metody osiągnęły skuteczność na poziomie powyżej 99%, z drobnymi różnicami między podejściami. Żadne z podejść nie było w stanie w pełni wyodrębnić klas mniejszościowych, może to być spowodowane faktem, że sama sieć GRU nie radziła sobie z tym zadaniem najlepiej w odosobnionych testach. Sprawdzone zaproponowane nowe podejścia jednak poprawiły wyniki architektury GRU w stosunku do wyników GRU działającej w standardowy sposób. W przyszłych badaniach planuje się wdrożyć tak przygotowaną metodę w rzeczywiste użycie w jednym z projektów, gdzie metoda będzie mogła sprawdzić się na trudniejszych, zawierających więcej szumu danych.

4.12. Wnioski i dalsze badania

W powyższej części pracy zaprezentowano wachlarz sposobów na modyfikację IDS opartych o uczenie maszynowe, przybliżono sposób ich działania i zaprezentowano szereg wyników eksperymentalnych, włącznie z testami nowej architektury opartej o ekstrakcję cech przy pomocy GRU i klasyfikację przez RF.

Wiele produktów opartych o ML w IDS pozwala na osiągnięcie wyników porównywalnych lub lepszych od wyników osiąganych przez ekspertów w tej dziedzinie używanych jest jako model "*black-box*", nie dając żadnych wskazówek co do powodów podjętych decyzji. Dobrym kierunkiem dalszych badań byłyby sposoby uzyskania wyjaśnienia wyników osiąganych przez algorytmy uczenia maszynowego w kontekście cyberbezpieczeństwa,

Tablica 23. Wyniki dla punktu odniesienia - ekstraktor GRU i klasyfikator RF, bez balansowania i PCA, accuracy: 0.9900

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	436364
Bot	0.72	0.87	0.79	482
DDoS	0.98	0.97	0.97	38691
DoS GoldenEye	0.80	0.91	0.85	2707
DoS Hulk	0.99	0.99	0.99	69311
DoS Slowhttptest	0.96	0.95	0.95	1667
DoS slowloris	0.98	0.99	0.98	1724
FTP-Patator	0.99	0.99	0.99	2388
Heartbleed	0.67	1.00	0.80	2
PortScan	1.00	1.00	1.00	47632
SSH-Patator	0.98	0.99	0.99	1746
Brute Force	0.28	0.54	0.37	236
Sql Injection	0.00	0.00	0.00	0
XSS	0.06	0.22	0.09	50
macro avg	0.74	0.81	0.77	603000
weighted avg	0.99	0.99	0.99	603000

Tablica 24. Zbiór CICIDS2017 zbalansowany przez Random Subsampling, ekstrakcja cech przez GRU, klasyfikacja RF, accuracy: 0.9918

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	166922
Bot	0.82	0.87	0.84	558
DDoS	0.99	0.99	0.99	38433
DoS GoldenEye	0.87	0.95	0.91	2843
DoS Hulk	0.99	0.99	0.99	69356
DoS Slowhttptest	0.98	0.97	0.98	1666
DoS slowloris	0.98	0.99	0.99	1725
FTP-Patator	1.00	1.00	1.00	2377
Heartbleed	0.67	1.00	0.80	2
PortScan	1.00	1.00	1.00	47633
SSH-Patator	0.98	0.98	0.98	1767
Brute Force	0.67	0.63	0.65	480
Sql Injection	0.00	0.00	0.00	0
XSS	0.29	0.38	0.33	150
macro avg	0.80	0.84	0.82	333912
weighted avg	0.99	0.99	0.99	333912

Tablica 25. Zbiór CICIDS2017 zbalansowany przez Random Subsampling, ekstrakcja cech przez GRU, PCA przed klasyfikatorem, klasyfikacja RF, accuracy: 0.9927

precision	recall	f1-score	support	
BENIGN	0.99	0.99	0.99	166944
Bot	0.84	0.87	0.86	569
DDoS	0.99	0.99	0.99	38426
DoS GoldenEye	0.89	0.95	0.92	2903
DoS Hulk	0.99	0.99	0.99	69249
DoS Slowhttptest	0.99	0.97	0.98	1676
DoS slowloris	0.98	0.99	0.98	1723
FTP-Patator	1.00	1.00	1.00	2378
Heartbleed	1.00	1.00	1.00	3
PortScan	1.00	1.00	1.00	47630
SSH-Patator	0.98	0.98	0.98	1778
Brute Force	0.66	0.63	0.64	476
Sql Injection	0.00	0.00	0.00	2
XSS	0.24	0.31	0.27	155
macro avg	0.83	0.83	0.83	333912
weighted avg	0.99	0.99	0.99	333912

Tablica 26. Zbiór CICIDS2017, ekstrakcja cech przez GRU, klasyfikacja cost-sensitive RF, accuracy: 0.9903

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	435436
Bot	0.80	0.76	0.78	618
DDoS	0.97	0.97	0.97	38612
DoS GoldenEye	0.79	0.91	0.85	2681
DoS Hulk	0.99	0.99	0.99	69278
DoS Slowhttptest	0.96	0.90	0.93	1758
DoS slowloris	0.93	0.99	0.96	1637
FTP-Patator	0.99	0.99	0.99	2389
Heartbleed	0.67	1.00	0.80	2
PortScan	1.00	1.00	1.00	47636
SSH-Patator	0.98	1.00	0.99	1743
Brute Force	0.73	0.30	0.43	1088
Sql Injection	0.00	0.00	0.00	0
XSS	0.11	0.17	0.13	122
macro avg	0.78	0.78	0.77	603000
weighted avg	0.99	0.99	0.99	603000

Tablica 27. Zbiór CICIDS2017, ekstrakcja cech przez GRU, PCA przed klasyfikatorem, klasyfikacja cost-sensitive RF, accuracy: 0.9903

	precision	recall	f1-score	support
BENIGN	0.99	0.99	0.99	435436
Bot	0.80	0.76	0.78	618
DDoS	0.97	0.97	0.97	38612
DoS GoldenEye	0.79	0.91	0.85	2681
DoS Hulk	0.99	0.99	0.99	69278
DoS Slowhttptest	0.96	0.90	0.93	1758
DoS slowloris	0.93	0.99	0.96	1637
FTP-Patator	0.99	0.99	0.99	2389
Heartbleed	0.67	1.00	0.80	2
PortScan	1.00	1.00	1.00	47636
SSH-Patator	0.98	1.00	0.99	1743
Brute Force	0.73	0.30	0.43	1088
Sql Injection	0.00	0.00	0.00	0
XSS	0.11	0.17	0.13	122
macro avg	0.78	0.78	0.77	603000
weighted avg	0.99	0.99	0.99	603000

co mogłoby podnieść zaufanie w sztuczną inteligencję i zagwarantować wartość dodaną dla pracujących z nimi operatorów *Security Operations Center* (SOC).

5. Część druga: Zagadnienie detekcji ataków z grupy adversarial

5.1. Zjawisko ataków z grupy adversarial - przegląd literatury

Najnowsze osiągnięcia w dziedzinie uczenia maszynowego w połączeniu ze wzrostem mocy obliczeniowej najnowszych komputerów przyczyniły się do znacznego rozprzestrzenienia się zastosowań technologii opartych o sztuczną inteligencję w wielu dziedzinach dzisiejszego życia. Spostrzeżenia uzyskane dzięki wykorzystaniu uczenia maszynowego z zebranych danych drastycznie usprawniły takie dziedziny jak służba zdrowia, finanse, jak i systemy bezpieczeństwa [84].

Uczenie maszynowe posiada zdolność odnajdywania zależności pomiędzy cechami w dużych zbiorach danych. Pojawił się szereg metod ML, takich jak SVM, *clustering*, sieci neuronowe itp. Procedury ML zazwyczaj podążają za tym samym ogólnym wzorem - najpierw ma miejsce faza uczenia, po której następuje faza testów, w której ML wykonuje klasyfikację, bądź regresję.

To w fazie uczenia algorytm dostosowuje model do danych, wprowadzonych zazwyczaj z dużego zbioru. Algorytmy ML osiągają zadziwiająco dobre rezultaty w szerokiej gamie zastosowań [7].

W chwili obecnej większość algorytmów proponowanych przez badaczy, naukowców i specjalistów z branży Research and Development skupia się jedynie na kwantyfikowalnej jakości uzyskanych rezultatów - na wysokiej wydajności i niskiej liczbie błędów (typu false positive i false negative). Niemniej jednak, nawet kiedy te cele zostają osiągnięte, modele te nie mogą - a może raczej nie powinny - być implementowane w warunkach realnych, zwłaszcza w domenach krytycznych, czy aspektach życia, które mogą mieć wpływ na całe społeczeństwa, bez uwzględnienia innych kryteriów i wymagań dotyczących sztucznej inteligencji. Są nimi: bezpieczeństwo algorytmów, ich wyjaśnialność i uczciwość.

Co więcej, w chwili obecnej niesamowite rezultaty osiągane są na danych, które są odpowiednio przygotowane w warunkach laboratoryjnych i osiągalne są jedynie gdy implementacja też zachodzi w takich warunkach.

Niemniej jednak, zastosowanie AI na wielką skalę stało się rzeczywistością, za czym idzie świadomość, że bezpieczeństwo samych algorytmów ML wymaga natychmiastowej uwagi. Wrodzy użytkownicy (ang. *adversaries*), potrafią starannie dobrać próbki wejściowe podawane do algorytmów AI w tak, że zmienia to wyniki klasyfikacji bądź regresji w wygodny dla nich sposób. Pomimo wszechobecności ML, świadomość zagrożeń związanych z ich użyciem i ich podatność na ataki typu adversarial jest jeszcze całkiem niewielka [36].

Obecne rozwiązania zaproponowane przez badania opierają się o takie zabiegi jak douczanie algorytmów, używanie defensywnej destylacji danych (ang. *defensive distillation*) albo opierające się o generatywne sieci współzawodniczące (ang. *Generative Adversarial Network* - GAN). Zabiegi te działają na konkretne rodzaje ataków, ale nie zabezpieczają równo przed wszystkimi. Stosowanie części z nich powoduje również pogorszenie wyników osiąganych przez ML [21].

Zadaniem tego działu będzie przybliżenie najnowszych prac związanych z bezpieczeństwem ML, ich ilustracja znajduje się na rys. 17.

Poruszone zostaną najnowsze zagrożenia związane z algorytmami AI, takie jak ataki typu *evasion*, *poisoning* i *exploratory*.

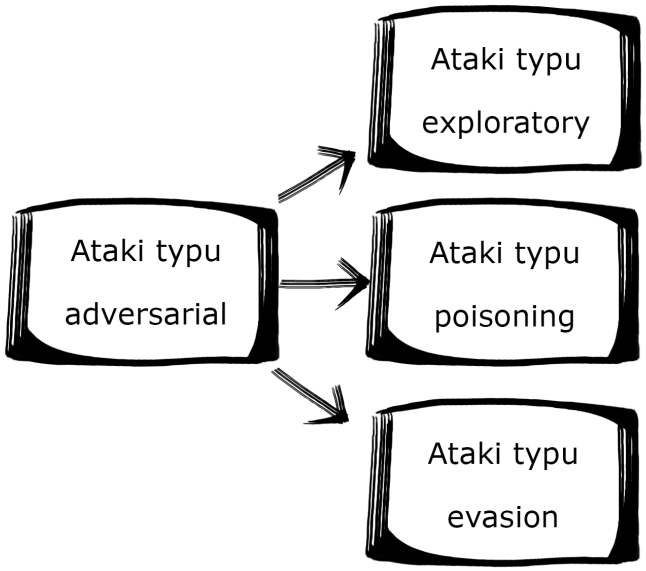
5.2. Ataki typu poisoning

Niedawno zwrócono uwagę, że starannie przygotowane dane wejściowe mogą wpłynąć na algorytmy sztucznej inteligencji by przechylić wyniki klasyfikacji w sposób odpowiadający zamiarom atakującego [22].

Taka nowa przeszkoda w popularyzowaniu metod uczenia maszynowego nie była do tej pory szczegółowo zbadana, w związku z czym świadomość tego zjawiska jest proporcjonalnie niewielka. Niemniej jednak w chwili pisania tej pracy odkryto cały wachlarz zagrożeń bezpieczeństwa z tej kategorii[22].

W wyniku niedawnego drastycznego wzrostu zainteresowania dziedziną zabezpieczania algorytmów ML powstał szereg różnych propozycji dotyczących metod obronnych. Badania nad zabezpieczeniem uczenia maszynowego trwają, zaproponowano już kilka rozwiązań, ale nie istnieje jeszcze w pełni odporny system i nie ma jeszcze rozwiązań sprawdzonych w praktyce [65].

Pierwszym z omawianych ataków jest *poisoning*, czyli "zatruwanie" danych. W literaturze znaleźć można kilka różnych ataków tego typu. W [10] przedstawiona jest metoda wykorzystująca sposób działania algorytmu SVM. Ogólnym założeniem ataku jest wprowadzenie do zbioru wykorzystywanego do uczenia algorytmu próbki która znacząco zmieni wyniki



Rysunek 17. Nowe ataki na uczenie maszynowe

klasyfikacji, obniżając osiągnięcia uzyskanego modelu. Taką próbkę można, według autorów, stworzyć poprzez rozwiązanie odpowiedniego problemu optymalizacyjnego.

Do znalezienia lokalnych maksymów powierzchni funkcji błędu wykorzystali oni algorytm *gradient ascent*. Artykuł opiera się o odwracanie etykiet konkretnych próbek w klasyfikacji binarnej, określany przez autorów jako wrogi szum etykiet (ang. *adversarial label noise*). W swojej pracy porównują podany powyżej sposób do efektów uzyskanych przez odwracanie etykiet dokonywane przypadkowo.

Dane w zbiorze użytym do weryfikacji nie są w żaden sposób zmieniane. Autorzy zwracają uwagę, że odnalezienie najlepszego miksu odwracania etykiet nie jest rzeczą łatwą [10].

W [78] zawarta została ilustracja w jaki sposób atakujący może manipulować algorytmem uczenia maszynowego w systemie filtrowania spamu, tak by doprowadzić do wyświetlenia się reklamy u użytkownika, lub zmusić filtr do wyeliminowania poprawnej korespondencji.

Celem badanej metody był algorytm znany jako SpamBayes. SpamBayes bierze pod uwagę tak nagłówek jak i treść wiadomości, zamienia słowa na tokeny i nadaje wiadomości punktację w celu klasyfikacji jako "*spam*", "*ham*" albo "*unsure*". Artykuł prezentuje sposób wykonania ataku

przy pomocy słownika, w którym algorytm poddany jest szeregowi spamowych wiadomości zawierających konkretny zestaw słów, które mają wysokie prawdopodobieństwo wystąpienia w prawdziwej wiadomości. Gdy te są oznaczane jako spam, algorytm uczy się oznaczać wiadomości zawierające te zwroty jako spam.

Ten konkretny atak ma dwie wersje: procedurę w której wiadomość po prostu zawiera cały słownik, atak znany jako "*basic dictionary attack*", albo bardziej wyrafinowany sposób, gdzie e-mail zawiera dystrybucję słów która ma większe prawdopodobieństwo by znaleźć się we wiadomościach od konkretnego użytkownika, wraz z kolokwializmami, literówkami itp. W tym konkretnym przypadku zaproponowano zbiór postów z forum Usenet.

Drugim rozpatrywanym atakiem jest metoda służąca do blokowania konkretnego rodzaju wiadomości - atak określony mianem "przyczynowy celowany atak na dostępność" (ang. *causative targeted availability attack*) lub "atak skupiony" (ang. *focused attack*). W tym scenariuszu atakujący zalewa algorytm serią wiadomości zawierających konkretny ciąg znaków. Gdy SpamBayes przeprowadza procedurę aktualizacji algorytmu - douczając go - dołącza te wiadomości do zbioru wykorzystywanego w uczeniu modelu i staje się podatny na klasyfikowanie wiadomości zawierających ten konkretny ciąg znaków jako spam.

Atak ten może być użyty do wyeliminowania wiadomości konkurencyjnej firmy, np. przy przetargach lub konkursach. Dołączenie nazwy konkurującej firmy do spamowych wiadomości, nazwy ich produktów albo imion ich pracowników mogłoby osiągnąć właśnie taki cel. Autorzy artykułu wskazują, że używanie ataków słownikowych może spowodować, że użytkownik w ogóle wyłączy filtr antyspamowy, ponieważ jego istnienie staje się bezcelowe gdy atakujący przejmie kontrolę nad zaledwie 1% zbioru wykorzystywanego do douczenia algorytmu. Mistrzowsko wykonany atak skupiony jest w stanie według autorów skierować konkretną wiadomość do katalogu spam w 90% przypadków.

W artykule [125] umieszczono propozycję podstawowych zasad ewaluacji bezpieczeństwa algorytmów doboru cech. Zasady te podążają za szkicem omówionym w [9], gdzie autorzy biorą pod lupę cel atakującego, zakres wiedzy jaką posiada o sposobie działania atakowanego algorytmu oraz zakres w jakim są w stanie wpłynąć na dane. Zadaniem wrogiego użytkownika jest albo atak celowany, albo masowy (ang. *indiscriminate*) z zamiarem przekroczenia jednej z trzech cech dobrze znanej w świecie ochrony informacji triady: dostępności (ang. *avaiability*), spójności (ang. *integrity*) i prywatności (ang. *privacy*).

Dokładny poziom znajomości sposobu działania atakowanego algorytmu posiadany przez atakującego można wyrazić w jeden z następujących sposobów:

- wiedza o zbiorze danych do uczenia (częściowa lub pełna)
- wiedza o reprezentacji cech (częściowa lub pełna)
- wiedza o algorytmie doboru cech
- wiedza całkowita (najgorszy możliwy scenariusz)
- wiedza niepełna

Rozpatrując przypadek pod względem zdolności atakującego (ang. *attacker's capability*) w wypadku ataków typu *poisoning*, zazwyczaj atakujący jest w stanie wpłynąć tylko na część zbioru wykorzystywanego do uczenia algorytmu. W różnych przypadkach ataków typu poisoning atakujący musi brać pod uwagę różne scenariusze nadawania etykiet próbkom, na przykład przy wykorzystaniu honeypotów i programów antywirusowych, stanowiąc pewne ograniczenie dla tworzonych próbek.

W [100] autorzy badają atak typu poisoning stworzony tak, by oszukać nawet ludzkiego operatora nadającego etykiety, bądź sprawdzającego zgodność etykiet. Nazywają to "*clean-label attacks*", czyli ataki o czystej etykiecie.

Co więcej, ich praca nie zakłada wiedzy na temat zbioru wykorzystywanego do uczenia algorytmu, wymaga jednak pewnej wiedzy o samym algorytmie. Atak oparty jest o optymizację i można go przeprowadzić tak dla sieci opartych o transfer wiedzy (ang. *transfer-learning*) jak i dla sieci wytrenowanych od początku do końca.

Ogólna procedura ataku, który autorzy nazywają "Trujące Żaby" (ang. *Poison Frogs*) wygląda następująco: najbardziej podstawowa wersja ataku zaczyna się od wybrania próbki docelowej, następnie zmieniania tej próbki, aż do chwili w której wygląda jakby należała do klasy bazowej.

Tak przygotowana próbka, będąca atakiem typu *poisoning* jest wstrzykiwana do zbioru. Zadanie jest uważane z wykonane gdy próbka, która była wzięta na cel ataku zostaje sklasyfikowana jako klasa bazowa w chwili testu.

Wytworzenie odpowiedniej próbki, która ma zostać wstrzyknięta do zbioru wykorzystywanego do uczenia algorytmu odbywa się poprzez proces określany jako "kolizja cech" (ang. *feature collision*). Jest to procedura, która wykorzystuje nieliniową złożoność funkcji propagującej dane wejściowe poprzez przedostatnią warstwę sieci neuronowej w celu znalezienia próbki która ma podobne cechy do próbki obranej za cel, ale znajduje się również blisko klasy bazowej.

Dobranie ataku w taki sposób pozwala oszukać także ewentualnego operatora, który mógłby sprawdzać dane przed dodaniem ich do zbioru wykorzystywanego do uczenia. Sama optymizacja wykonywana jest przy pomocy procedury "*iterative forward-backward-splitting*".

W [76] ewaluowana jest procedura ataku typu *poisoning* celująca w modele wieloklasowe oparte o *gradient-descent*. Autorzy wykorzystują nowozaproponowaną optymizację *back-gradient*. Metoda ta oferuje mniej wymagający obliczeniowo i bardziej niezawodny sposób uzyskania rozwiązania zadania optymalizacyjnego prowadzącego do stworzenia ataków typu *poisoning*. Czerpiąc z modeli opartych o energię oraz z optymalizacji hiperparametrów, to podejście pozwala na zamianę jednego z problemów optymalizacji na zestaw iteracji poprawiania parametrów.

Autorzy przedstawiają procedurę ataku typu *poisoning* której celem są głębokie sieci neuronowe. Demonstrują swoją metodę na konwolucyjnej sieci neuronowej (CNN) nauczonej przy pomocy zbioru ręcznie pisanych cyfr MNIST[61]. Podsumowując, odkrywają że głębokie sieci zdają się być bardziej odporne od zwykłych algorytmów ML, przynajmniej w warunkach zatruwania poniżej 1% danych.

Przeprowadzony zostaje również eksperyment polegający na zastosowaniu ataków typu *poisoning* stworzonych dla głębokich sieci neuronowych w innych modelach i odwrotnie. Autorzy podsumowują badanie wynikiem, że ataki stworzone dla regresji liniowej (ang. *Linear Regression* LR) nie są skuteczne przeciwko CNN, natomiast ataki przeciwko CNN mają podobną skuteczność przeciwko LR jak losowe odwrócenia etykiet.

W [130] zaproponowano sposób ominięcia kalkulacji gradientu poprzez częściowe wykorzystanie koncepcji GAN. W tym podejściu zastosowano autoenkoder w celu stworzenia zatrutych próbek, funkcja kosztu decyduje o punktacji wyniku. Tak stworzone dane podaje się sieci neuronowej, a gradient wysyłany jest z powrotem do generatora.

Efektywność tej metody jest gruntownie sprawdzona na zbiorach MNIST [61] i CIFAR-10 [55] - znanych zbiorach obrazów. Wybrano następującą architekturę: sieć o dwóch ukrytych warstwach osiągająca skuteczność na poziomie 96,82% na zbiorze MNIST. Dla zbioru CIFAR-10 wykorzystano konwolucyjną sieć neuronową o dwóch warstwach konwolucyjnych i dwóch warstwach w pełni połączonych. Skuteczność tego modelu to 71,2%. Dla celów demonstracyjnych wstrzykiwany jest jeden atak na raz. Autorzy podsumowują, że wykorzystanie GAN jest ulepszeniem w stosunku do wcześniejszych metod i nadaje się do ataku na głębokie sieci i zbiory wykorzystywane do uczenia ich.

Inny atak zaproponowany jest w [24], noszący nazwę "celowanego ataku backdoor" (ang. *targeted backdoor attack*). Założeniem tej metody jest

stworzenie “ukrytego wejścia” w systemie autoryzacyjnym opartym o algorytm sztucznej inteligencji, pozwalając atakującym przejść przez proces uwierzytelniania, oszukując go. Zatrute próbki stworzone są w taki sposób by kategoryzować konkretne cechy jako etykietę obraną przez atakującego. Autorzy proponują metodę działającą na relatywnie małych próbkach ataków, zakładając, że atakujący nie ma żadnej wiedzy o użytym algorytmie sztucznej inteligencji. Ich metoda poparta jest przykładem jak wstrzyknięcie tylko 50 próbek uzyskuje 90% skuteczności.

5.3. Ataki typu exploratory

Zwięźle ujmując, ataki typu “exploratory” są sposobem na zbudowanie funkcjonalnego ekwiwalentu wdrożonego już modelu opartego o ML, ekstrahując granicę decyzyjną, ustawienia algorytmu, jego własności, jak i informacje na temat użytych danych uczenia [132].

Tak, jak w wypadku ataków typu *poisoning*, poziom zagrożenia jaki stanowi atakujący jest uzależniony od jego wiedzy o atakowanym modelu. Zakres zachowań wrogiego użytkownika jest określony jego poziomem dostępu do modelu. [84, 21].

Poziom zaznajomienia z celem ataku można ogólnie skategoryzować jako “*white-box*” i “*black-box*”.

Ataki *black-box* są podejmowane bez jakiejkolwiek wcześniejszej wiedzy o modelu i jakiejkolwiek jego części. Ataki tego typu mogą być przeprowadzone przez staranne podawanie danych wejściowych do modelu i obserwowanie sparowanych z nimi danych wyjściowych. Wiedzę z jednego modelu można do pewnego stopnia przetransferować pomiędzy wieloma różnymi algorytmami [84].

Ataki typu exploratory można podzielić na następujące podkategorie:

- *Non-Adaptive Black-Box Attack* (nie dostosowujący się atak black-box) - w celu wytworzenia ataków typu evasion na lokalnej kopii atakowanego modelu wrogi agent kradnie klasyfikator. W tej kategorii ataków atakujący ma dostęp tylko do dystrybucji danych wykorzystanych do uczenia algorytmu i stosuje je do wykonania lokalnej kopii modelu [21].
- *Adaptive Black-Box Attack* (adaptacyjny atak black-box) - w tym przypadku atakujący nie ma żadnej wiedzy o algorytmie ale może przeprowadzić procedurę ataku podobną do “*chosen-plaintext*” z dziedziny kryptografii, gdzie wrogi agent wysyła do atakowanego modelu serię starannie przygotowanych zapytań [21] [15].

– *Strict Black-Box Attack* - w tym wypadku atakujący może zebrać pary danych wejściowych i wyjściowych z klasyfikatora, ale nie jest w stanie przeprowadzić ataku w sposób adaptacyjny. Takie podejście może odnieść sukces gdy zebrana zostanie duża liczba takich par[21].

W wypadku ataków *black-box* atakujący może zacząć procedurę wykradania modelu bez znajomości zbioru danych wykorzystanych do uczenia algorytmu. Może to uczynić wysyłając zestaw próbek, dla których dostanie etykiety wyjściowe z modelu. Następnie używając par próbek wejściowych i etykiet które zwrócił model, może nauczyć algorytm opierający się o wielopoziomową sieć neuronową, tworząc funkcjonalną kopię atakowanego modelu.

W ramach badań wykonanych na potrzeby tej pracy na algorytm uczony tak uzyskanym zbiorem danych wybrano głęboką sztuczną sieć neuronową, ponieważ architektura ta wykazuje wyjątkową umiejętność dopasowywania się do danych [31]. Co więcej, [132] pokazuje jak można użyć sieci DNN w celu zbudowania funkcjonalnego ekwiwalentu klasyfikatorów opartych o algorytm *Naïve Bayes* i SVM. W dodatku, w [104] można znaleźć opis jak przy pomocy DNN wykraść klasyfikator już wdrożony warunkach rzeczywistych poprzez wysyłanie zapytań na API, korzystając z darmowej licencji i zbioru danych zebranych w sieci. Celem ataku typu *exploratory* jest budowa lokalnej wersji klasyfikatora [21].

Ekstrakcja modelu (model extraction) to po prostu kradzież, która za nic ma sobie poufność konkretnego modelu ML, pozwalając atakującemu na stworzenie surogatu atakowanego algorytmu[93].

Ataki *white-box* mogą być przeprowadzone na cały szereg sposobów. Architektura algorytmów może mieć wiele różnych postaci w zależności od dobranych hiperparametrów. Wiedza o konkretnych wartościach tych parametrów może zostać użyta do znalezienia konkretnych czułych punktów, które atakujący może później wykorzystać. Dobrze poinformowany, wrogo nastawiony użytkownik może wpłynąć na dane wejściowe tak, by skierować algorytm w miejsce, w którym osiąga niezadowalające rezultaty [84].

W scenariuszu *white-box* zakłada się, że atakujący ma pełną wiedzę o rodzaju, sposobie działania i konkretnych ustawieniach algorytmu, który atakuje, włącznie z liczbą ukrytych warstw, liczbą neuronów na tych warstwach, doborze hiperparametrów takich jak optymizator. Zakłada się również, że zna parametry wdrożonego modelu. Ta wiedza jest przydatna w znajdowaniu miejsc w których algorytm jest podatny na atak. Wiedza o wagach modelu może być zamieniona w druzgoczący atak [21].

Podgrupy ataków typu extraction:

– Odwrócenie Modelu (ang. *Model Inversion*),

– Atak Wnioskujący Członkostwo (ang. *Membership Inference attack*),
– Ekstrakcja Modelu przez API (ang. *Model Extraction via APIs*),
– Wnioskowanie Informacji (ang. *Information Inference*) [21].

5.4. Ataki typu evasion

Wykrywanie ataków sieciowych jest krytycznym aspektem zapewniania cyberbezpieczeństwa. Co jednak się stanie, jeśli system IDS sam stanie się celem ataku? Innymi słowy, co chroni system chroniący? Poniższa części niniejszej pracy dotyczyć będzie problemu ataków z grupy *adversarial*, konkretnie typu *evasion* i sposobom na zabezpieczenie systemów wykrywania ataków sieciowych i ich komponentów opartych o ML. Założeniem jest propozycja metody wykrywania ataków typu *evasion*. Obecnie wykorzystywane algorytmy uczenia maszynowego nie były tworzone do działania we wrogich środowiskach w których są teraz implementowane.

Z tego powodu rozwiązania oparte o ML stały się ostatnio celem szeregu ataków. Poniższe działy rozważają możliwość obniżenia skuteczności działania dobrze zoptymalizowanego systemu wykrywania ataków sieciowych w chwili testu, poprzez stworzenie ataków z grupy adversarial, przy pomocy czterech ataków dopiero co zaproponowanych w społeczności naukowej, następnie opracowanie metody wykrywającej te nowe ataki.

Według najlepszej wiedzy autora wykrywanie ataków z grupy adversarial na sztuczne sieci neuronowe nie było jeszcze badane w kontekście systemów wykrywania ataków sieciowych (IDS).

Wykorzystanie metod ML w domenie cyberbezpieczeństwa cieszy się coraz większą popularnością, uczenie maszynowe przenika również do wielu innych dziedzin życia. W tym dziale poruszono inny temat krytyczny dla działania systemów wykrywania ataków sieciowych opartych o uczenie maszynowe i działania ML jako takiego.

Poruszony tutaj problem zyskał na popularności w ostatnim czasie. Intensywne badania nad rozwiązaniami opartymi o ML i popularność głębokiego uczenia sprawiły, że na powierzchnię wypłynął szereg wcześniej nie omawianych problemów. Jednym z najbardziej palących wyzwań jest zjawisko znane jako "*adversarial attacks*". W ciągu ostatnich kilku lat algorytmy oparte o uczenie się z danych, które stanowią podstawę działania całego wachlarza inteligentnych systemów, same stały się celem ataków.

Przypadek ataków z grupy adversarial, wyciągnięty na światło dzienne przez [126], w szybkim tempie staje się poważnym zagrożeniem dla nowoczesnych rozwiązań opartych o sztuczną inteligencję, zwłaszcza w związku z popularnością tych technologii w krytycznych sferach życia, jak samoprowadzące się samochody, biometria czy cyberbezpieczeństwo [124].

W następnych rozdziałach rozważona zostanie możliwość zaatakowania sztucznej sieci neuronowej wykorzystywanej do wykrywania ataków sieciowych. Atak przeprowadzony zostanie przy pomocy technik określanych jako "*evasion attack*". Następnie przedstawiona zostanie innowacyjna propozycja rozwiązania które będzie wykrywać ataki tego typu.

5.4.1. Generowanie ataków typu adversarial zaadaptowanych z metod rozpoznawania obrazów na algorytmy uczenia maszynowego wykorzystywane wykrywaniu ataków sieciowych

Na przestrzeni ostatnich kilku lat badania nad swoistymi cechami algorytmów ML stały się zdecydowanie bardziej popularne. Fakt, że starannie przygotowany wektor cech może oszukać nawet te klasyfikatory, które na porównawczym zbiorze danych osiągają lepsze wyniki niż ludzie skupił na sobie uwagę społeczności naukowej zajmującej się sztuczną inteligencją. Wraz ze wzrostem świadomości problemu pojawiły się badania odkrywające kolejne czułe punkty algorytmów AI [21]. Ataki z grupy adversarial to próbki, które dla ludzkiego oka wyglądają prawie identycznie do próbek które klasyfikują się poprawnie. Niemniej jednak, mała, świadomie dobrana zmiana, będąca jednocześnie jednym z najgorszych możliwych przypadków dla klasyfikatora, wprowadzona do próbki podanej do nauczonego modelu może spowodować, że cały szereg algorytmów ML, w tym, co najważniejsze, sztuczne sieci neuronowe, zaklasyfikują tę próbkę niepoprawnie [110]. Poniżej przedstawiono cztery sposoby na stworzenie takiego wektora cech.

Atak typu "Szybkie Oznaczanie Gradientu" (ang. Fast Gradient Sign Method)

Autorzy [36] znaleźli metodę na tworzenie działających ataków z grupy adversarial, które potrafią wywołać defekt klasyfikacji w szeregu algorytmów ML. Metodę początkowo zademonstrowano na zbiorach ImageNet, MNIST [61] i CIFAR-10 [55], będących znanymi, porównawczymi zbiorami obrazów.

Podejście polega na znalezieniu małego wektora szumu. Zsumowanie tego wektora koresponduje ze znakiem elementów gradientu funkcji kosztu badanej próbki. Metodę *Fast Gradient Sign* można zdefiniować jako linearyzację funkcji kosztu wokół obecnej wartości Θ, otrzymując optymalne zaburzenie ograniczone w sposób max-norm z poniższego równania:

$$\eta = \epsilon sign(\nabla_x J(\Theta, x, y)) \tag{9}$$

Gdzie Θ to parametry modelu, x dane wejściowe podawane do modelu, y to cele odpowiednich x i $J(\Theta, x, y)$ jest funkcją kosztu wykorzystywaną do nauczenia sieci [36]. W literaturze do tej metody odnosi się jako do "*Fast Gradient Sign Method*" (FGSM), "*Fast Gradient Method*" (FGM) lub po prostu *Fast Method*.

Atak typu "Podstawowa Metoda Iteracyjna" (ang. Basic Iterative Method)

Autorzy [58] oferują rozszerzenie metody FGM polegające na stosowaniu jej wielokrotnie przy małej wartości każdego kroku, obcinając wartości po każdym kroku, używając $\alpha = 1$, co oznacza zmienianie wartości każdego elementu (np. piksela) o 1. W swojej pracy autorzy określali liczbę iteracji heurystycznie, tak, by sięgnąć granicy max-norm ϵ. Formuła wygląda w ten sposób:

$$X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_N^{adv} + \alpha sign(\nabla_x J(X_N^{adv}, y_{true}))\} \tag{10}$$

Atak typu Carlini and Wagner

W [19] autorzy proponują rozwiązanie kwestii tworzenia ataków poprzez sformułowanie problemu optymalizacji w taki sposób, że może być rozwiązany przez algorytmy optymalizacyjne znane w dniu dzisiejszym. Problem optymalizacyjny jest formalnie zdefiniowany jako:

$$\begin{aligned} zminimalizuj \quad & D(x, x+\delta) \\ tak\,by \quad & C(x+\delta) = t \\ & x + \delta \in [0,1]^n \end{aligned} \tag{11}$$

Gdzie x się nie zmienia, więc szukane jest δ które minimalizuje $D(x, x+\delta)$. Innymi słowy - szukane jest δ które zmieni wynik klasyfikacji. D jest miarą odległości; w swoim artykule autorzy rozpatrują trzy z nich - L_0, L_2 and L_∞. Na potrzeby tej pracy wybrano L_2.

W celu przebudowania formuły tak, by mogła zostać rozwiązana , autorzy przedefiniowują funkcję celu f tak, że $C(x+\delta) = t$ gdy $f(x+\delta) \leq 0$ i proponują szereg sposobów rozwiązania równania f. W niniejszej pracy spośród ataków zdefiniowanych w [19] wybrano L_2.

Atak typu "Projekcja Gradientu Prostego" (ang. Projected Gradient Descent)

W [67] autorzy proponują metodę nazwaną Projected Gradient Descent, jako najsilniejszy atak z grupy adversarial i uniwersalne zagrożenie najwyższego stopnia. Jak autorzy to ujmują, jest to ostateczna metoda wykonująca ograniczoną optymalizację na dużą skalę (constrained large-scale optimization). Pokrótce, wspomniany wcześniej FGM jest metodą tworzenia ataków opierającą się na jednym kroku. W teorii bardziej niebezpieczną opcją byłaby procedura polegająca na wielu krokach, którą autorzy określają jako "w sumie to projected gradient descent". Atak określa równanie (12).

$$X^{t+1} = \Pi_{x+S}(x^t + \alpha sgn(\nabla_x L(\Theta, x, y))) \tag{12}$$

Generowanie ataków - podsumowanie

Idąc w ślad za podsumowaniem umieszczonym w [67], model ataku można ostatecznie sformułować jako dwupoziomowy problem optymalizacyjny, wyrażony przez równanie (13):

$$\begin{aligned} &\min_{\theta} \rho(\Theta), \\ gdzie\, \rho(\Theta) = E_{(x,y)\sim D}[&\max_{\delta \in S} L(\Theta, x+\delta, y)] \end{aligned} \tag{13}$$

Gdzie S jest zbiorem dopuszczalnych wzburzeń (ang. *perturbations*), D oznacza dystrybucję, L jest funkcją kosztu, a $E_{(x,y)\sim D}$ jest zmienioną próbką podawaną klasyfikatorowi. Takie sformułowanie pozwala na rozważenie problemów *wewnętrznej maksymalizacji* oraz *zewnętrznej minimalizacji*. W istocie powyżej opisane cztery ataki są różnymi podejściami rozwiązania tak określonego problemu.

5.4.2. Poziom wiedzy atakującego o atakowanym systemie

Poziom zagrożenia, który stanowi atakujący ograniczony jest zakresem informacji, które atakujący jest w stanie zebrać o sposobie działania atakowanego algorytmu. Wiedza ta wpływa z kolei na wachlarz ataków, których może się podjąć. W literaturze ten poziom zaznajomienia określa się mianem *Adversarial Capabilities*, co można by luźno przetłumaczyć jako "wrogie zdolności". Kategoryzuje się je na *black-box* i *white box* [84, 21].

W najprostszych możliwych słowach - atakujący z kategorii black-box nie mają żadnej wiedzy o algorytmie, który jest ich celem, w odróżnieniu

od kategorii white-box, gdzie atakujący mają pełną wiedzę o algorytmie, stanowiąc najsilniejszy możliwy przykład atakującego [84].

W tej pracy przyjęto model atakującego zakładający jego pełną wiedzę o algorytmie.

W rzeczywistej sytuacji do pozyskania odpowiedniej wiedzy na temat algorytmu można skorzystać z innego rodzaju ataku, określanego jako ekstrakcja modelu (model extraction). Procedura ta polega na wykradnięciu wiedzy zgromadzonej w działającym modelu, którą można wykorzystać do stworzenia lokalnej kopii modelu o porównywalnych właściwościach, co wystarczy do wyprodukowania ataków typu evasion [53]. Kwestia ta jest dokładnie przeanalizowana w innym dziale tej pracy.

5.4.3. Przeciwdziałanie atakom z grupy adversarial

Zaproponowano szereg możliwych mechanizmów obronnych przed atakami z grupy adversarial. Jednym z takich mechanizmów jest "*adversarial retraining*", polegający na próbie poprawnego sklasyfikowania tych ataków poprzez dołączenie próbek ataków do zbioru przeznaczonego do uczenia klasyfikatora jako początkowa klasa lub poprzez stworzenie oddzielnej klasy dla takich ataków [110, 36, 64, 38].

Taka metoda ma swoje zalety, ale nie jest jednak efektywna przeciwko nowym rodzajom ataków i częstokroć powoduje pogorszenie wyników tworzonego modelu.

Innym sposobem radzenia sobie z problemem ataków z grupy adversarial jest metoda *Defensive Distillation*. Zaproponowana w [85], metoda ta początkowo służyła do stworzenia głębokiej sieci neuronowej przy pomocy wiedzy uzyskanej z innej DNN, transferując wiedzę do mniejszej architektury [43]. Metoda ta pozyskuje pomocnicze obserwacje o próbkach użytych do uczenia pod postacią prawdopodobieństw wystąpienia klas, które są wykorzystywane jako dodatkowe cechy w trakcie uczenia. Metoda pozwala na stworzenie gładszych modeli klasyfikacyjnych, redukując ich podatność na ataki z grupy adversarial [85]. Ataki metodą Carlini and Wagner pozwalają jednak pokonać tą procedurę [19].

Istnieje grupa badaczy proponująca stworzenie osobnego klasyfikatora w celu wykrywania ataków z grupy adversarial [35]. Badacze Ci twierdzą, że metoda ta jest odporna na ataki stworzone przy pomocy FGM i Jacobian Sailency Map Attack [83]. Podejście to wykorzystuje tę samą dystrybucję danych, która została użyta do nauczenia klasyfikatora, przez co można na nim wykonać atak typu evasion przez przeformułowanie ataku w sposób uwzględniający drugi klasyfikator, tak jak ukazuje to [18].

W odróżnieniu od metod odnalezionych w literaturze podejście z niniejszej pracy w celu wykrycia ataków zbiera wszystkie aktywacje neuronów ze wszystkich warstw architektury - schemat podejścia umieszczono na rys. 22. Jest to możliwe, ponieważ budowa sieci wykorzystywanej od wykrywania ataków sieciowych może wykorzystywać stosunkowo niewielką liczbę warstw, osiągając zadowalające wyniki. Według najlepszej wiedzy autora takie podejście do wykrywania ataków typu evasion celujących w systemy wykrywania ataków sieciowych nie zostało jeszcze odnotowane w literaturze.

5.4.4. Wnioski

Według literatury [21, 103, 110] nie wypracowano jeszcze w pełni bezpiecznych rozwiązań, ani nie przetestowano ich w jeszcze "w terenie" [66]. Metody wypracowane do chwili obecnej mają zastosowanie do konkretnych rodzajów ataków. Niektóre z tych metod prowadzą do pogorszenia wyników zastosowanych metod ML [21].

Dziedzina wykrywania i ochrony przed atakami z grupy adversarial jest więc w pilnej potrzebie badań. Większość istniejących metod skupia się na systemach rozpoznawania obrazu [19, 21, 36, 38, 58, 64, 66, 67, 68, 72, 85, 103, 110, 132], ataki z grupy adversarial są jednak skuteczne we wszystkich domenach w których stosuje się algorytmy uczenia maszynowego - w tym w systemach wykrywania ataków sieciowych [44]. Logiczną kontynuacją takiego rozumowania byłaby adaptacja znanych z dziedziny rozpoznawania obrazu technik obronnych do domeny wykrywania ataków sieciowych, jak i opracowanie nowych sposobów wykrywania tych ataków dla tej dziedziny, co będzie dalszą częścią niniejszej pracy.

6. Propozycja własnego detektora ataków typu evasion

6.1. Atak typu exploratory poprzez ekstrakcję modelu - metoda

Celem tego działu jest propozycja metody wykradzenia klasyfikatora w kontekście cyberbezpieczeństwa - poprzez podawanie do wdrożonego algorytmu uczenia maszynowego pakietu danych i nauczenie głębokiej sieci neuronowej przy pomocy tych danych i zaobserwowanych odpowiedzi z modelu. Taki atak jest pierwszym krokiem do wykonywania bardziej zaawansowanych ataków, jak ataki typu poisoning lub, jak w wypadku niniejszej pracy, typu evasion.

Posiadanie skopiowanej (lokalnej) wersji klasyfikatora jest w takim razie cenną własnością dla wrogiego użytkownika. W ramach niniejszej pracy ukazany zostanie proces wykradania modelu w warunkach laboratoryjnych. Model stworzony został przy pomocy sztucznej sieci neuronowej i jednego z wzorcowych zbiorów danych w cyberbezpieczeństwie. Szczegółowe ustawienia tego algorytmu omówione są w późniejszej sekcji. Niemniej jednak, by maksymalnie przybliżyć warunki eksperymentu do warunków rzeczywistych przyjęto zestaw założeń.

W rzeczywistej sytuacji związanej z cyberbezpieczeństwem jedyną możliwą do zaobserwowania odpowiedzią byłoby odmówienie połączenia użytkownikowi wykazującemu pewien zestaw zachowań. Jest to w pewnym sensie podobne do ataku znanego jako "*oracle attack*" w dziedzinie kryptografii [15]. W związku z tym wyekstrahowany model będzie w stanie wykonywać jedynie binarną klasyfikację.

Celem porówania stworzonego modelu z pierwotnym modelem, ten drugi też będzie binarnym klasyfikatorem, rozróżniającym pomiędzy ruchem normalnym i wykazującym znamiona anomalii.

Dla eksperymentu wykorzystano znany, wzorcowy zbiór danych.

6.1.1. Klasyfikator użyty w systemie wykrywania ataków sieciowych

Klasyfikatorem wykorzystanym w eksperymencie jest wielowarstwowa sztuczna sieć neuronowa nauczona przy pomocy wzorcowego zestawu danych NSL-KDD [112].

Zwyczajem w budowaniu architektury sztucznych sieci neuronowych, który podyktowany jest sposobem działania ANN, jest założenie liczby neuronów na pierwszej ukrytej warstwie; mniejszej, niż liczba cech z danych wejściowych [3, 31]. Jest to jednak istotna wskazówka dla atakującego. Rzeczywista liczba ukrytych warstw i neuronów na nich podyktowana jest jednak osiągami uzyskanego modelu [31].

Użycie takiej liczby neuronów w warstwach ukrytych, która jest mniejsza niż liczba neuronów w warstwie wejściowej owocuje pewnym uogólnieniem w reprezentacji, co może poprawić wyniki modelu po jego wdrożeniu [3].

Wykorzystana sieć złożona była z dwóch warstw ukrytych po 25 neuronów każda, ReLU w roli funkcji aktywacji i ADAM jako optymizator.

6.1.2. Ekstrakcja modelu

Architektura głębokiej sieci neuronowej użytej do ekstrakcji modelu

Jak podaje [3], płytka sztuczna sieć neuronowa zbudowana z jednej warstwy, ale z odpowiednio dużą liczbą neuronów byłaby, w teorii, w stanie znaleźć każdą funkcję wyznaczoną z danych. To samo źródło sugeruje też, że pogłębienie sieci o większa liczbę warstw jest opłacalną alternatywą (choć nie bez swoich własnych problemów).

Jako architekturę wykorzystaną do ekstrakcji modelu użyto wielowarstwową sztuczną sieć neuronową zbudowaną z 4 ukrytych warstw po 512 neuronów każda, oraz ReLU w roli funkcji aktywacji. Wybór taki zmotywowany był faktem, że klasyfikator oparty o głębokie uczenie osiągnął najlepsze wyniki w ekstrakcji modelu z systemów opartych o SVM oraz Naïve Bayes w [132].

Architektura została dobrana tak, by być bardziej złożona niż pierwotny klasyfikator, biorąc pod uwagę typową liczbę cech zawartą w zbiorach danych w dziedzinie cyberbezpieczeństwa.

Tak, jak zilustrowano w [132], algorytm ekstrakcji modelu można podsumować w niniejszy sposób:

Algorytm ekstrakcji modelu:

1. wysyłanie zapytań do klasyfikatora, podając mu dane wejściowe
2. obserwacja zwracanych przez klasyfikator etykiet
3. wykorzystanie par danych z wejścia/wyjścia w celu nauczenia głębokiej sieci neuronowej i optymalizacji jej hiperparametrów.

Proces jest również ukazany na rys. 6.1.3.

6.1.3. Opis przeprowadzonych eksperymentów

W celu należytego sprawdzenia sprawności modelu nauczonego przy pomocy etykiet pozyskanych z innego algorytmu, eksperyment przygotowano w następujący sposób:

Po pierwsze, zbiór danych został przekształcony na klasyfikację binarną. By to osiągnąć, wszystkie klasy zawarte w zbiorze zostały odpowiednio przekształcone na "attack" lub "benign". Szczegóły tej konwersji podane są w tab. 28.

Po przekształceniu zbiór zawiera 58628 próbek z kategorii "attack" i 67342 próbki z kategorii "benign". Stanowi to około 46,5% próbek "attack" oraz 53,5% próbek "benign".

Po drugie, zbinaryzowany zbiór NSL-KDD został podzielony na trzy części:

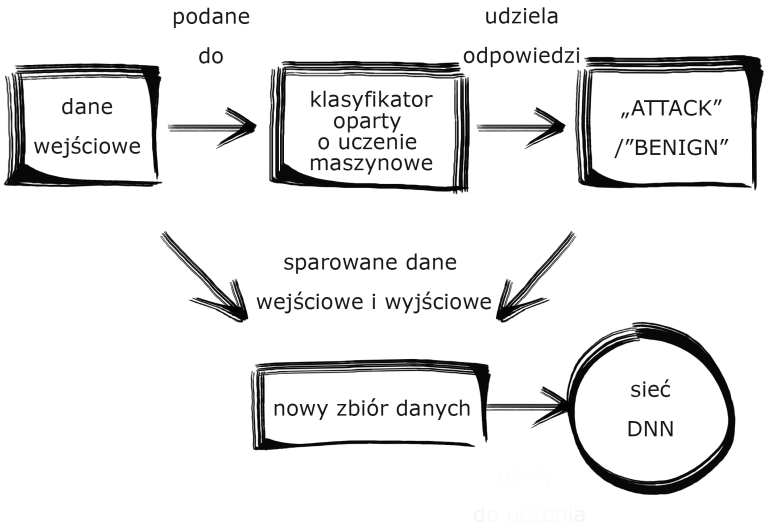
- Zbiór A - użyty do nauczenia pierwotnego modelu
- Zbiór B - użyty do podawania zapytań do pierwotnego modelu w celu uzyskania etykiet klasyfikacji, następnie do nauczenia sieci DNN
- Zbiór C - wykorzystany do weryfikacji sieci DNN i porównania wyników z pierwotnym modelem (tab.29 oraz tab.30)

Po trzecie, do pierwotnego modelu wysyłane są zapytania z próbek pobranych ze zbioru B, odpowiedzi z modelu są zapisywane, parowane z odpowiednimi próbkami i formowane w nowy zbiór danych. Ten nowy zbiór jest następnie wykorzystany do nauczenia algorytmu sieci DNN.

Ostatecznie zbiór C wykorzystany jest do weryfikacji uzyskanego modelu opartego o sieć DNN, próbki ze zbioru C są również podawane do pierwotnego modelu w celu porównania wyników z nowouzyskanym modelem.

6.2. Efekt ekstrakcji modelu i znaczenie eksperymentu

W tab. 29 umieszczone zostały wyniki klasyfikacji pierwotnego modelu. Klasa "*benign*" oznacza próbki nie zawierające znamion ataku, "*attack*"



Rysunek 18. Proces Ekstrakcji Modelu

odnosi się do próbek zawierających znamiona ataku. Jak widać w tablicy pomyłek, pierwotny klasyfikator popełnił 76 błędów z kategorii *false positive* i 78 błędów z kategorii *false negative* na całe 37791 klasyfikacji. Wynika z tego skuteczność na poziomie 99,59%

W tab. 30 znajdują się wyniki wyekstrahowanego modelu. Można zauważyć, że popełnił zaledwie 120 błędów typu false positive i 95 błędów typu false negative we wszystkich 37791 klasyfikacjach. Daje to skuteczność równą 99,43%, wynik porównywalny z pierwotnym modelem.

Wyżej omówione wyniki wskazują na to, że w warunkach laboratoryjnych możliwe jest osiągnąć podobne rezultaty co atakowany model poprzez

Tablica 28. Mapowanie klas z NSL-KDD do attack i benign

Klasy w NSL-KDD w eksperymencie	Klasy
“ipsweep”, “multihop”, “rootkit”, “warezclient”,	attack
“guess_passwd”, “phf”, “nmap”,	attack
“teardrop”, “neptune”, “loadmodule”, “imap”, “portsweep”	attack
, “pod”, “ftp_write”, “warezmaster”,	attack
“back”, “land”, ’smurf’, ’satan’, ’buffer_overflow’	attack
“normal”	benign

Tablica 29. Macierz konfuzji pierwotnego modelu na zbiorze C

	benign	attack
benign	20125	78
attack	76	17512

Tablica 30. Macierz konfuzji ekstrahowanego modelu na zbiorze C

	benign	attack
benign	20108	95
attack	120	17468

wysyłanie zapytań do wdrożonego modelu i używając uzyskanych etykiet jako podstawy do nauczenia głębokiej sieci.

Powyżej opisana i przetestowana procedura jest pierwszym krokiem na drodze do wytworzenia bardziej dotkliwych ataków z grupy adversarial. W nadchodzących działach ukazany zostanie sposób wykonania ataku typu "Evasion", następnie przedstawiona zostanie nowatorska propozycja metody wykrywania takich ataków.

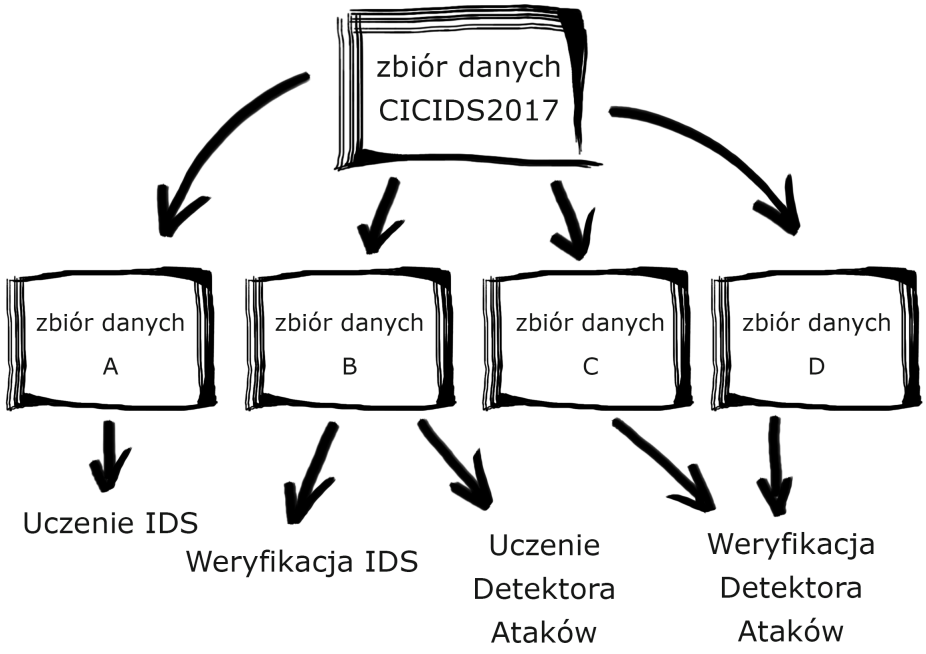
6.3. Proponowana metoda wykrywania ataków typu evasion w sztucznych sieciach neuronowych

W niniejszej sekcji zaprezentowane zostanie ogólne podejście do kwestii wykrywania ataków typu evasion na sztuczne sieci neuronowe. W pierwszej kolejności opisany zostanie proces wstępnej obróbki danych w przebiegu uczenia systemu IDS. Następnie wykonywane i testowane są ataki na system IDS. Zbierane są aktywacje neuronów powstające przy podawaniu tych ataków do systemu IDS, jak i aktywacje neuronów powstające przy podawaniu niezainfekowanych danych. Tak stworzony zbiór zostanie wykorzystany do nauczenia metody wykrywania ataków typu evasion w oparciu o aktywacje neuronów.

6.3.1. Wykrywanie ataków sieciowych oparte o sztuczną sieć neuronową

W tej sekcji przedstawiony zostanie proces uczenia IDS przy wykorzystaniu zbioru danych CICIDS2017 [102].

Zbiór został podzielony na cztery części wykorzystując stratyfikację, by wszystkie ataki wykrywane przez IDS znalazły się w każdym podzbiorze. Wynikiem tej procedury są następujące podzbiory:

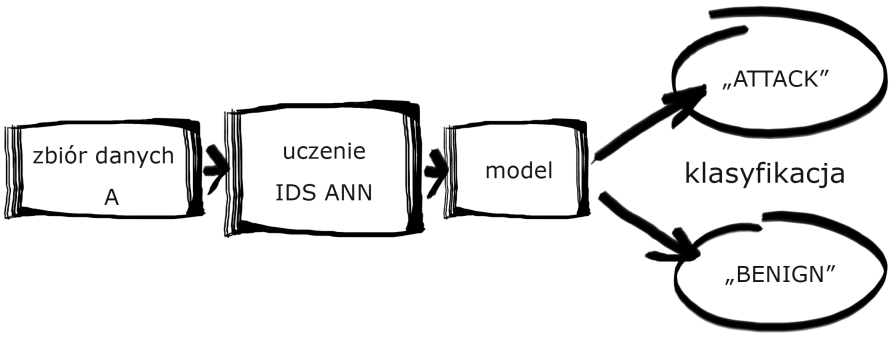


Rysunek 19. Wykorzystanie i podział CICIDS2017 na uczenie i weryfikację IDS ANN oraz Detektora Ataków typu Evasion

- Dataset A - wykorzystany do nauczenia klasyfikatora IDS
- Dataset B - wykorzystany do weryfikacji skuteczności klasyfikatora IDS oraz do stworzenia ataków typu evasion, następnie do pozyskania wartości aktywacji neuronów klasyfikatora IDS przy wprowadzaniu danych z klasy "benign", "attack" i "adversarial", które zostaną wykorzystane do nauczenia detektora ataków typu evasion.
- Dataset C i D - wykorzystane do stworzenia ataków typu evasion i pozyskania aktywacji neuronów systemu IDS potrzebnych do weryfikacji detektora.

Wszystkie podzbiory przystosowano do zadania klasyfikacji binarnej pozostawiając etykietę "BENIGN" dla klasy nie zawierającej ataków, zamieniając jednak wszystkie etykiety konkretnych podklas ataków na jedną zbiorczą etykietę "ATTACK". Wykorzystanie i podział CICIDS2017 jest zilustrowane na rys.19.

Architektura systemu IDS ANN wygląda następująco: skompilowana została sztuczna sieć neuronowa o trzech ukrytych warstwach, 40 neuronów na pierwszej warstwie, 40 na drugiej i 20 na trzeciej. Jako funkcję



Rysunek 20. Proces tworzenia IDS ANN

Tablica 31. “IDS ANN” uczony zbiorem A i testowany zbiorem B - wyniki testów

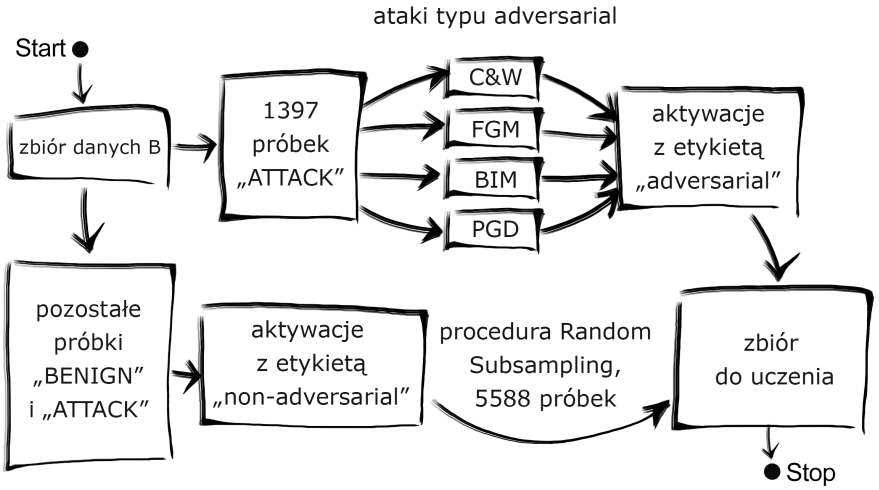
	precision	recall	f1-score	support
ATTACK	0.96	0.97	0.97	139675
BENIGN	0.99	0.99	0.99	405383
micro avg	0.98	0.98	0.98	545058
macro avg	0.97	0.98	0.98	545058
weighted avg	0.98	0.98	0.98	545058
samples avg	0.98	0.98	0.98	545058

aktywacji wykorzystano ReLU, użyty optymizator to “ADAM”. Batch size ustawiono na 100 a liczbę epoch na 10. Tak stworzona sieć uzyskała wynik 0.9827 w skuteczności po nauczeniu jej na Dataset A i weryfikacji na Dataset B. Metryki precision, recall i f1 ukazane są w tab. 31. Proces uczenia i weryfikacji IDS zilustrowany jest na rys. 20. Jak widoczne na ilustracji, zbinaryzowany zbiór podawany jest opisanej powyżej architekturze, ta poprzez proces uczenia, buduje model zdolny do wydajnej klasyfikacji.

6.3.2. Przeprowadzenie ataków typu evasion

Po weryfikacji nauczonego systemu IDS ANN (kwestię optymizacji sztucznych sieci neuronowych poruszono wcześniej, jak i w [88]) przygotowano cztery różne rodzaje ataków typu evasion w oparciu o próbki z klasy “ATTACK” pobrane z Dataset B. Do przygotowania tych ataków wykorzystano poniższe metody:

Ze zbioru B losowo wybrano 1397 próbek klasy “ATTACK” i wykorzystano jako bazę do stworzenia ataków typu evasion. W teście bez ataków



Rysunek 21. Tworzenie, na podstawie zbioru B, zbioru wykorzystanego do nauczenia detektora ataków typu evasion

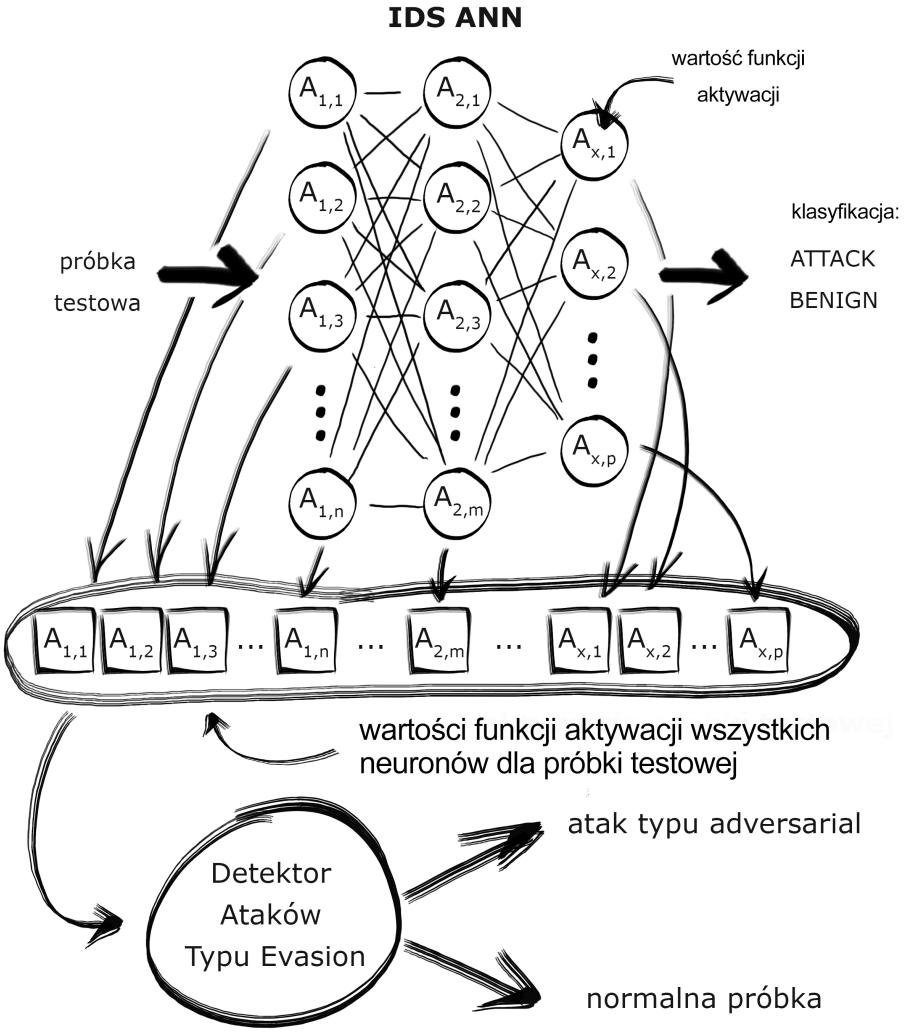
typu evasion IDS ANN sklasyfikował 1353 z nich jako ataki i 44 jako niegroźne. Wykorzystując powyższe 4 metody udało się oszukać algorytm i zmusić go do sklasyfikowania 1296ciu próbek z kategorii "attack" jako próbki niegroźne dla metod BIM i PGD, 1324 dla FGM i 59 dla CW.

Wymienione procedury wprowadzają nieprzypadkowy, wrogi szum (adversarial noise) do próbek. Ten szum w niektórych przypadkach owocował ujemnymi wartościami w niektórych cechach. Te ujemne wartości zamieniono na zera w celu utrzymania realistycznych warunków testu, to jednak w pewnym stopniu wpłynęło na efektywność ataków. IDS ANN sklasyfikował 55 więcej ataków jako próbki niegroźne w wypadku metod BIM i PGD, 295 więcej dla ataku CW i 21 mniej dla FGM.

Próbkom ze zbioru B niewykorzystanym przy tworzeniu ataków typu evasion nadano etykietę "nonadversarial", wytworzonym atakom nadano etykietę "adversarial". Przy 5588 próbkach "adversarial" pobrano losowo tyle samo próbek "nonadversarial" w celu stworzenia zbalansowanego zbioru dla nauczenia detektora ataków typu evasion. Procedura ta jest zilustrowana na rys.21. Dataset D poddany został tej samej procedurze, za wyjątkiem balansowania, w celu stworzenia zbioru do weryfikacji detektora.

– Carlini and Wagner attack (CW) [19]
– Fast Gradient Sign Method (FGM) [36]

– Basic Iterative Method (BIM) [58]
– Projected Gradient Descent (PGD) [67]



Rysunek 22. Uzyskiwanie wartości aktywacji neuronów IDS ANN dla konkretnych próbek testowych

6.3.3. Metoda Detekcji

Oba zbiory z atakami typu evasion i próbkami nie noszącymi znamion ataków zostały podane do IDS ANN i zebrano aktywacje wszystkich 102

neuronów, dla każdej z próbek, tak jak jest to zilustrowane na rys. 22. Próbkom nadano odpowiednie etykiety - “adversarial” dla aktywacji pobranych z próbek ataków i “nonadversarial” dla aktywacji uzyskanych z próbek nie będących atakami typu Evasion.

Aktywacje neuronów pobrane z próbek należących do zbilansowanego zbioru wykorzystano do nauczenia detektora ataków typu evasion.

Architektura detektora wygląda następująco: sztuczna sieć neuronowa o 3 ukrytych warstwach i ReLU jako funkcji aktywacji, 51 neuronów na pierwszej warstwie, 51 na drugiej i 25 na trzeciej. Optymalizator: ADAM. Proces uczenia i weryfikacji ukazano na rys.23. Przy wykorzystaniu batch size: 100 i 10 epoch detektor uzyskał 0.8506 skuteczności na zbiorze testowym. Szczegółowe wyniki zebrano w tab. 32.

7. Analiza wyników detektora ataków typu evasion

Jak pokazano w tab. 32 detektor uzyskuje wysokie wyniki skuteczności i *recall* dla klasy ataków typu evasion. Takie wyniki oznaczają, że proponowana metoda jest obiecująca.

W trakcie budowy detektora ataków typu evasion powstał zbiór danych wartości aktywacji neuronów ANN IDS. To pozwoliło na przetestowanie podejścia przy wykorzystaniu innych niż ANN klasyfikatorów. W tab. 33 umieszczono wyniki wykrywania w oparciu o algorytm Random Forest (RF). Jak się okazało, metoda ta uzyskała wyniki lepsze od ANN we wszystkich 3 metrykach, zwłaszcza warta uwagi jest skuteczność, która przekroczyła 91% (91.24).

Stosując się do wyżej opisanego podejścia wypróbowano również inną metodę należącą do kategorii *ensemble*. Algorytm ADABoost nie poprawił wyników osiągniętych przez Random Forest, osiągając jedynie 87.66% skuteczności, wyniki te były nadal lepsze od tych osiągniętych przez ANN. Szczegółowe zestawienie wyników ADABoost można znaleźć w tab. 34.

Następnie rozszerzono testy o klasyfikator SVM. Jak wynika z metryk umieszczonych w tab. 35, SVM nie udało się wyłonić wzorów wskazujących na obecność ataków typu evasion w oparciu o zbiór aktywacji neuronów i osiągnął zaledwie 0,79 w metryce recall.

Tablica 32. Wyniki dla detektora ataków typu evasion opierającego się o sieć ANN przy wykorzystaniu zbioru testowego

	precision	recall	f1-score	support
adversarial	0.06	0.91	0.11	5588
non-adversarial	1.00	0.85	0.92	543661
micro avg	0.85	0.85	0.85	549249
macro avg	0.53	0.88	0.51	549249
weighted avg	0.99	0.85	0.91	549249
samples avg	0.85	0.85	0.85	549249

Tablica 33. Wyniki osiągane przez detektor ataków typu evasion oparty o algorytm Random Forest przy wykorzystaniu aktywacji neuronów ze zbioru testowego

	precision	recall	f1-score	support
adversarial	0.11	0.99	0.20	5588
non-adversarial	1.00	0.91	0.95	543661
micro avg	0.92	0.91	0.92	549249
macro avg	0.56	0.95	0.58	549249
weighted avg	0.99	0.91	0.95	549249
samples avg	0.91	0.91	0.91	549249

Tablica 34. Wyniki osiągane przez detektor ataków typu evasion oparty o algorytm ADABoost przy wykorzystaniu aktywacji neuronów ze zbioru testowego

	precision	recall	f1-score	support
adversarial	0.07	0.90	0.13	5588
non-adversarial	1.00	0.88	0.93	543661
macro avg	0.53	0.89	0.53	549249
weighted avg	0.99	0.88	0.93	549249

Tablica 35. Wyniki osiągane przez detektor ataków typu evasion oparty o algorytm SVM przy wykorzystaniu aktywacji neuronów ze zbioru testowego

	precision	recall	f1-score	support
adversarial	0.11	0.79	0.19	5588
non-adversarial	1.00	0.93	0.97	543661
macro avg	0.55	0.86	0.58	549249
weighted avg	0.99	0.93	0.96	549249

Rysunek 23. Proces uczenia i testowania detektora ataków typu evasion

Tablica 36. Wyniki osiągane przez detektor ataków typu evasion oparty o algorytm Nearest Neighbour przy wykorzystaniu aktywacji neuronów ze zbioru testowego

	precision	recall	f1-score	support
adversarial	0.11	0.99	0.20	5588
nonadv	1.00	0.91	0.95	543661
macro avg	0.56	0.95	0.58	549249
weighted avg	0.99	0.91	0.95	549249

W ostatnim kroku ten sam zbór aktywacji neuronów wykorzystano do nauczenia klasyfikatora k-NN. Metoda ta pozwoliła osiągnąć wyniki podobne do wyników osiąganych przez klasyfikator Random Forest. Szczegóły umieszczono w tab.36.

8. Wnioski

Niniejsza rozprawa podzielona była na dwie części, w których zaproponowano dwie innowacyjne metody wykrywania ataków sieciowych przy pomocy algorytmów opartych o techniki uczenia maszynowego. Pierwsza, będąca modyfikacją metod uczenia maszynowego w zastosowaniu IDS i druga, będąca propozycją metody wykrywającej ataki skierowane przeciwko samym mechanizmom uczenia maszynowego będących trzonem systemów IDS.

W części pierwszej pracy ukazano jaki wpływ na algorytmy ML mają metody balansowania danych, jak można podnieść wyniki tej samej architektury przez odpowiednie zestawienie hiperparametrów oraz zaaplikowano nową architekturę Gated Recurrent Unit do wykrywania ataków sieciowych. Wszystkie te procedury odbyły się na zapisie ruchu sieciowego zawierającego relewantne dla dnia dzisiejszego zapisy prawdziwych ataków sieciowych.

Eksperymenty pokazały innowacyjne metody przekraczające 99% skuteczności wykrywania ataków sieciowych dla wzorcowego zbioru danych NSL-KDD i 97% dla zbioru CICIDS2017 przy pomocy metody opartej o sztuczną sieć neuronową. Nie udało się poprawić wyników skuteczności wykrywania klas mniejszościowych przy pomocy różnych metod balansowania danych dla algorytmów opartych o sieci neuronowe; zauważono natomiast, że dla nowoczesnego zapisu ruchu sieciowego metoda random subsampling daje porównywalne lub lepsze wyniki od bardziej zaawansowanych podejść. Z sukcesem zaproponowano natomiast innowacyjne użycie sieci opartej o rekurencyjną sieć Gated Recurrent Unit i uzyskano satysfakcjonujące wyniki tak dla zbioru wzorcowego jak i dla zbioru zawierającego nowoczesne dane.

W części drugiej zaproponowano metodę wykrywania ataków z grupy "adversarial", typu "evasion". Innowacyjna metoda polegająca na badaniu wartości funkcji aktywacji neuronów przy pomocy odpowiednio nauczonego modelu opierającego się o sztuczną sieć neuronową i inne metody uczenia maszynowego pozwala wykryć 99% ataków typu "evasion", Badania jednak pokazały, że jest to okupione dużym odsetkiem błędów typu *false positive*.

W części drugiej zademonstrowano również sposób przeprowadzenia skutecznego ataku typu *exploratory* na wdrożony system IDS, co pozwala atakującemu na stworzenie jego kopii lokalnej.

W oparciu o przeprowadzone badania, ich wyniki i analizę można powziąć wniosek, że potwierdziła się teza pracy.

Bibliografia

[1] 2019. IBM X-Force Report. URL https://newsroom.ibm.com/.

[2] Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G.S., Davis A., Dean J., Devin M., Ghemawat S., Goodfellow I., Harp A., Irving G., Isard M., Jia Y., Jozefowicz R., Kaiser L., Kudlur M., Levenberg J., Mané D., Monga R., Moore S., Murray D., Olah C., Schuster M., Shlens J., Steiner B., Sutskever I., Talwar K., Tucker P., Vanhoucke V., Vasudevan V., Viégas F., Vinyals O., Warden P., Wattenberg M., Wicke M., Yu Y., Zheng X., 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[3] Aggarwal C.C., 2018. Neural Networks and Deep Learning A Textbook. Springer, Cham. ISBN 978-3-319-94462-3. doi:10.1007/978-3-319-94463-0.

[4] Alsulami B., Mancoridis S., 2018. Behavioral Malware Classification using Convolutional Recurrent Neural Networks. In: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), pp. 103–111. doi:10.1109/MALWARE.2018.8659358.

[5] Althubiti S., Nick W., Mason J., Yuan X., Esterline A., 2018. Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection. In: SoutheastCon 2018, pp. 1–5. ISSN 1558-058X. doi:10.1109/SECON.2018.8478898.

[6] Andrysiak T., Łukasz Saganowski, Chorań M., Kozik R., 2014. Network Traffic Prediction and Anomaly Detection Based on ARFIMA Model. In: Advances in Intelligent Systems and Computing, pp. 545–554. Springer International Publishing. doi:10.1007/978-3-319-07995-0_54. URL https://doi.org/10.1007%2F978-3-319-07995-0_54.

[7] Ateniese G., Felici G., Mancini L.V., Spognardi A., Villani A., Vitali D., 2013. Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. CoRR abs/1306.4447. URL http://arxiv.org/abs/1306.4447.

[8] Bielec A. Ąnalysis of a Polish BankBot"URL https://www.cert.pl/en/news/single/analysis-of-a-polish-bankbot/.

[9] Biggio B., Fumera G., Roli F., 2014. Pattern recognition systems under attack: Design issues and research challenges. International Journal of Pattern Recognition and Artificial Intelligence 28(07), p. 1460002.

[10] Biggio B., Nelson B., Laskov P., 2012. Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389 .

[11] Bilge L., Dumitraş T., 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM conference on Computer and communications security, pp. 833–844.

[12] Bobowska B., Choraś M., Woźniak M., 2018. Advanced Analysis of Data Streams for Critical Infrastructures Protection and Cybersecurity. J. UCS 24(5), pp. 622–633.

[13] Breiman L., 2001. Random Forests. Machine Learning 45(1), pp. 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[14] C. Aggarwal C., 2018. Neural Networks and Deep Learning: A Textbook. ISBN 978-3-319-94462-3. doi:10.1007/978-3-319-94463-0.

[15] Cachin C., Camenisch J., 2004. Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings 3027. doi:10.1007/b97182.

[16] Canfora G., Di Sorbo A., Mercaldo F., Visaggio C.A., 2015. Obfuscation Techniques against Signature-Based Detection: A Case Study. In: 2015 Mobile Systems Technologies Workshop (MST), pp. 21–26. doi:10.1109/MST.2015.8.

[17] Cardenas A.A., Amin S., Sastry S., 2008. Secure Control: Towards Survivable Cyber-Physical Systems. In: 2008 The 28th International Conference on Distributed Computing Systems Workshops. IEEE. doi:10.1109/icdcs.workshops.2008.40. URL https://doi.org/10.1109%2Ficdcs.workshops.2008.40.

[18] Carlini N., Wagner D., 2017. Adversarial Examples Are Not Easily Detected. Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security - AISec '17 doi:10.1145/3128572.3140444. URL http://dx.doi.org/10.1145/3128572.3140444.

[19] Carlini N., Wagner D., 2017. Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE.

[20] Çamtepe S.A., Yener B., 2007. Modeling and detection of complex attacks. In: Proc. of the 3rd Int. Conf. on Security and Privacy in Communications Networks, pp. 234–243. doi:10.1109/SECCOM.2007.4550338.

[21] Chakraborty A., Alam M., Dey V., Chattopadhyay A., Mukhopadhyay D., 2018. Adversarial Attacks and Defences: A Survey. CoRR abs/1810.00069. URL http://arxiv.org/abs/1810.00069.

[22] Chakraborty A., Alam M., Dey V., Chattopadhyay A., Mukhopadhyay D., 2018. Adversarial attacks and defences: A survey. arXiv preprint arXiv:1810.00069 .

[23] Chen P.Y., Yang S., McCann J.A., Lin J., Yang X., 2015. Detection of false data injection attacks in smart-grid systems. IEEE Communications Magazine 53(2), pp. 206–213.

[24] Chen X., Liu C., Li B., Lu K., Song D., 2017. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 .

[25] Cho K., Van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 .

[26] Chollet F., et al., 2015. Keras. https://github.com/fchollet/keras.

[27] Choraś M., Kozik R., 2015. Machine learning techniques applied to detect cyber attacks on web applications. Logic Journal of the IGPL 23(1), pp. 45–56. doi:10.1093/jigpal/jzu038.

[28] Choraś M., Kozik R., Puchalski D., Hołubowicz W., 2013. Correlation Approach for SQL Injection Attacks Detection. In: Advances in Intelligent Systems and Computing, pp. 177–185. Springer Berlin Heidelberg. doi:10.1007/978-3-642-33018-6_18. URL https://doi.org/10.1007%2F978-3-642-33018-6_18.

[29] Collobert R., Bengio S., Marithoz J., 2002. Torch: A Modular Machine Learning Software Library.

[30] Cup K., 2007. Available on: http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html.

[31] da Silva · Danilo Hernane Spatti Rogerio Andrade Flauzino Luisa Helena Bartocci Liboni Silas Franco dos Reis Alves I.N., 2017. Artificial Neural Networks A Practical Course. ISBN 978-3-319-43161-1. doi:10.1007/978-3-319-43162-8.

[32] Djenouri Y., Belhadi A., Lin J.C.W., Cano A., 2019. Adapted K-Nearest Neighbors for Detecting Anomalies on Spatio–Temporal Traffic Flow. IEEE Access 7, pp. 10015–10027. doi:10.1109/access.2019.2891933. URL https://doi.org/10.1109%2Faccess.2019.2891933.

[33] Godefroy E., Totel E., Hurfin M., Majorczyk F., 2014. Automatic generation of correlation rules to detect complex attack scenarios. In: 10th International Conference on Information Assurance and Security, pp. 23–28. doi:10.1109/ISIAS.2014.7064615.

[34] Gong W., Fu W., Cai L., 2010. A Neural Network Based Intrusion Detection Data Fusion Model. In: 2010 Third International Joint Conference on Computational Science and Optimization, volume 2, pp. 410–414. doi:10.1109/CSO.2010.62.

[35] Gong Z., Wang W., Ku W.S., 2017. Adversarial and Clean Data Are Not Twins.

[36] Goodfellow I.J., Shlens J., Szegedy C., 2014. Explaining and Harnessing Adversarial Examples.

[37] Goyal P., Pandey S., Jain K., 2018. Unfolding Recurrent Neural Networks, pp. 119–168. Apress, Berkeley, CA. ISBN 978-1-4842-3685-7. doi:10.1007/978-1-4842-3685-7_3. URL https://doi.org/10.1007/978-1-4842-3685-7_3.

[38] Grosse K., Manoharan P., Papernot N., Backes M., McDaniel P., 2017. On the (Statistical) Detection of Adversarial Examples.

[39] Haddadi F., Khanchi S., Shetabi M., Derhami V., 2010. Intrusion Detection and Attack Classification Using Feed-Forward Neural Network. In: 2010

Second International Conference on Computer and Network Technology, pp. 262–266. doi:10.1109/ICCNT.2010.28.

[40] Han H., Wang W.Y., Mao B.H., 2005. Borderline-SMOTE: A New Over-sampling Method in Imbalanced Data Sets Learning. In: Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I, ICIC'05, pp. 878–887. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-28226-2, 978-3-540-28226-6. doi:10.1007/11538059_91. URL http://dx.doi.org/10.1007/11538059_91.

[41] Han H., Wang W.Y., Mao B.H., 2005. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: D.S. Huang, X.P. Zhang, G.B. Huang, eds., Advances in Intelligent Computing, pp. 878–887. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-31902-3.

[42] Hao Y., Sheng Y., Wang J., 2019. Variant Gated Recurrent Units With Encoders to Preprocess Packets for Payload-Aware Intrusion Detection. IEEE Access 7, pp. 49985–49998. ISSN 2169-3536. doi:10.1109/ACCESS.2019.2910860.

[43] Hinton G., Vinyals O., Dean J., 2015. Distilling the Knowledge in a Neural Network.

[44] Ibitoye O., Shafiq O., Matrawy A., 2019. Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks. arXiv preprint arXiv:1905.05137 .

[45] Idika N., Mathur A., 2007. A survey of malware detection techniques. Purdue University .

[46] Jakhale A.R., 2017. Design of anomaly packet detection framework by data mining algorithm for network flow. In: 2017 International Conference on Computational Intelligence in Data Science(ICCIDS). IEEE. doi:10.1109/iccids.2017.8272665. URL https://doi.org/10.1109%2Ficcids.2017.8272665.

[47] James G. W.D.H.T..T.R., 2013. An introduction to statistical learning. In: Cluster Comput (2018).

[48] Kan Z., Wang H., Xu G., Guo Y., Chen X., 2018. Towards Light-Weight Deep Learning Based Malware Detection. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), volume 01, pp. 600–609. ISSN 0730-3157. doi:10.1109/COMPSAC.2018.00092.

[49] Kelion L. ęBay redirect attack puts buyers' credentials at risk"URL http://www.bbc.com/news/technology-29241563.

[50] Khan R.U., Zhang X., Alazab M., Kumar R., 2019. An Improved Convolutional Neural Network Model for Intrusion Detection in Networks. In: 2019 Cybersecurity and Cyberforensics Conference (CCC), pp. 74–77. ISSN null. doi:10.1109/CCC.2019.000-6.

[51] Kim J., Kim J., Thu H.L.T., Kim H., 2016. Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In: 2016 International Conference on Platform Technology and Service (PlatCon), pp. 1–5. doi:10.1109/PlatCon.2016.7456805.

[52] Kohavi R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai, pp. 1137–1145.

[53] Kozik R., Choraś M., 2016. Solution to Data Imbalance Problem in Application Layer Anomaly Detection Systems. pp. 441–450. ISBN 978-3-319-32033-5. doi:10.1007/978-3-319-32034-2_37.

[54] Kozik R. C.M., 2016. Solution to Data Imbalance Problem in Application. Martinez-Alvarez F., Troncoso A., Quintian H., Corchado E. (Eds.): Hybrid Artificial Intelligent Systems, LNAI vol. 9648, pp. 441–450. ISSN 1076-9757.

[55] Krizhevsky A., Nair V., Hinton G., 2014. The CIFAR-10 dataset. online: http://www. cs. toronto. edu/kriz/cifar. html 55.

[56] Kumar Singh Gautam R., Doegar E.A., 2018. An Ensemble Approach for Intrusion Detection System Using Machine Learning Algorithms. In: 2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence), pp. 14–15. ISSN null. doi:10.1109/CONFLUENCE.2018.8442693.

[57] Kunal, Dua M., 2019. Machine Learning Approach to IDS: A Comprehensive Review. In: 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 117–121. ISSN null. doi:10.1109/ICECA.2019.8822120.

[58] Kurakin A., Goodfellow I., Bengio S., 2016. Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533 .

[59] Lanoe D., Hurfin M., Totel E., 2018. A Scalable and Efficient Correlation Engine to Detect Multi-Step Attacks in Distributed Systems. In: IEEE 37th Symposium on Reliable Distributed Systems (SRDS), pp. 31–40. doi:10.1109/SRDS.2018.00014.

[60] Le T., Kang H., Kim H., 2019. The Impact of PCA-Scale Improving GRU Performance for Intrusion Detection. In: 2019 International Conference on Platform Technology and Service (PlatCon), pp. 1–6. doi:10.1109/PlatCon.2019.8668960.

[61] LeCun Y., 1998. The MNIST database of handwritten digits. http://yann.lecun. com/exdb/mnist/ .

[62] Lee D. "MyFitnessPal breach affects millions of Under Armour users"URL `http://www.bbc.com/news/technology-43592470`.

[63] Lei Y., 2017. Network Anomaly Traffic Detection Algorithm Based on SVM. In: 2017 International Conference on Robots & Intelligent System (ICRIS). IEEE. doi:10.1109/icris.2017.61. URL `https://doi.org/10.1109%2Ficris.2017.61`.

[64] Li B., Vorobeychik Y., Chen X., 2016. A General Retraining Framework for Scalable Adversarial Classification.

[65] Liao X., Ding L., Wang Y., 2011. Secure Machine Learning, a Brief Overview. In: 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion, pp. 26–29. IEEE.

[66] Liao X., Ding L., Wang Y., 2011. Secure Machine Learning, a Brief Overview. In: 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion, pp. 26–29. doi:10.1109/SSIRI-C.2011.15.

[67] Madry A., Makelov A., Schmidt L., Tsipras D., Vladu A., 2017. Towards Deep Learning Models Resistant to Adversarial Attacks.

[68] Madry A., Makelov A., Schmidt L., Tsipras D., Vladu A., 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv e-prints arXiv:1706.06083.

[69] Maimon O., Rokach L., 2010. Data Mining and Knowledge Discovery Handbook, 2nd ed. ISBN 9780387098227.

[70] Maimon O., Rokach L., 2010. Data Mining and Knowledge Discovery Handbook, 2nd ed. ISBN 9780387098227.

[71] McGraw G., Morrisett G., 2000. Attacking Malicious Code: A Report to the Infosec Research Council. IEEE Softw. 17(5), pp. 33–41. ISSN 0740-7459. doi:10.1109/52.877857. URL http://dx.doi.org/10.1109/52.877857.

[72] Metzen J.H., Genewein T., Fischer V., Bischoff B., 2017. On Detecting Adversarial Perturbations.

[73] Michel C., Mé L., 2001. ADeLe: an Attack Description Language for Knowledge-based Intrusion Detection. In: Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001), pp. 353–365.

[74] Morin B., Mé L., Debar H., Duccassé M., 2009. M4D4: a Logical Framework to Support Alert Correlation in Intrusion Detection. Information Fusion 10(4), pp. 285–299.

[75] Mukhopadhyay I., Chakraborty M., Chakrabarti S., Chatterjee T., 2011. Back propagation neural network approach to Intrusion Detection System. In: 2011 International Conference on Recent Trends in Information Systems, pp. 303–308. doi:10.1109/ReTIS.2011.6146886.

[76] Muñoz-González L., Biggio B., Demontis A., Paudice A., Wongrassamee V., Lupu E.C., Roli F., 2017. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security - AISec 17. ACM Press. doi:10.1145/3128572.3140451. URL https://doi.org/10.1145%2F3128572.3140451.

[77] Naseer S., Saleem Y., Khalid S., Bashir M.K., Han J., Iqbal M.M., Han K., 2018. Enhanced Network Anomaly Detection Based on Deep Neural Networks. IEEE Access 6, pp. 48231–48246. ISSN 2169-3536. doi:10.1109/ACCESS.2018.2863036.

[78] Nelson B., Barreno M., Chi F.J., Joseph A.D., Rubinstein B.I., Saini U., Sutton C.A., Tygar J.D., Xia K., 2008. Exploiting Machine Learning to Subvert Your Spam Filter. LEET 8, pp. 1–9.

[79] Nguyen K.D.T., Tuan T.M., Le S.H., Viet A.P., Ogawa M., Minh N.L., 2018. Comparison of Three Deep Learning-based Approaches for IoT Malware Detection. In: 2018 10th International Conference on Knowledge and Systems Engineering (KSE), pp. 382–388. doi:10.1109/KSE.2018.8573374.

[80] Nwankpa C., Ijomah W., Gachagan A., Marshall S., 2018. Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378 .

[81] ÖNEY M.U., PEKER S., 2018. The Use of Artificial Neural Networks in Network Intrusion Detection: A Systematic Review. In: 2018 International

Conference on Artificial Intelligence and Data Processing (IDAP), pp. 1–6. IEEE.

[82] Ozkan K., Isik S., Kartal Y., 2018. Evaluation of convolutional neural network features for malware detection. In: 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pp. 1–5. doi:10.1109/ISDFS.2018.8355390.

[83] Papernot N., McDaniel P., Jha S., Fredrikson M., Celik Z.B., Swami A., 2016. The Limitations of Deep Learning in Adversarial Settings. 2016 IEEE European Symposium on Security and Privacy (EuroSP) doi:10.1109/eurosp.2016.36. URL http://dx.doi.org/10.1109/EuroSP.2016.36.

[84] Papernot N., McDaniel P., Sinha A., Wellman M.P., 2018. SoK: Security and Privacy in Machine Learning. In: 2018 IEEE European Symposium on Security and Privacy (EuroSP), pp. 399–414. doi:10.1109/EuroSP.2018.00035.

[85] Papernot N., McDaniel P., Wu X., Jha S., Swami A., 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. 2016 IEEE Symposium on Security and Privacy (SP) doi:10.1109/sp.2016.41. URL http://dx.doi.org/10.1109/SP.2016.41.

[86] Pasero S.B.A.E.F.C.M.E., 2016. Advances in Neural Networks. ISBN 978-3-319-33746-3. doi:10.1007/978-3-319-33747-0.

[87] Pattewar T.M., Sonawane H.A., 2015. Neural network based intrusion detection using Bayesian with PCA and KPCA feature extraction. In: 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), pp. 83–88. doi:10.1109/CGVIS.2015.7449898.

[88] Pawlicki M., Kozik R., Choraś M., 2019. Artificial Neural Network Hyperparameter Optimisation for Network Intrusion Detection. In: Intelligent Computing Theories and Application - 15th International Conference, ICIC 2019, Nanchang, China, August 3-6, 2019, Proceedings, Part I, pp. 749–760. doi:10.1007/978-3-030-26763-6_72. URL https://doi.org/10.1007/978-3-030-26763-6_72.

[89] Pawlicki M., Marchewka A., Choraś M., Kozik R., 2019. Gated Recurrent Units for Intrusion Detection. In: Image Processing and Communications - Techniques, Algorithms and Applications, IP&C'2019, Bydgoszcz, Poland, 11-13 September 2019, Proceedings, pp. 142–148. doi:10.1007/978-3-030-31254-1_18. URL https://doi.org/10.1007/978-3-030-31254-1_18.

[90] Pearson K., 1901. LIII. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2(11), pp. 559–572.

[91] Peng W., Kong X., Peng G., Li X., Wang Z., 2019. Network Intrusion Detection Based on Deep Learning. In: 2019 International Conference on Communications, Information System and Computer Engineering (CISCE), pp. 431–435. IEEE.

[92] Pérez J.D.J.S., Rosales M.S., Cruz-Cortés N., 2016. Universal Steganography Detector Based on an Artificial Immune System for JPEG Images. In:

2016 IEEE Trustcom/BigDataSE/ISPA, pp. 1896–1903. ISSN 2324-9013. doi:10.1109/TrustCom.2016.0290.

[93] Quiring E., Arp D., Rieck K., 2018. Forgotten Siblings: Unifying Attacks on Machine Learning and Digital Watermarking. In: 2018 IEEE European Symposium on Security and Privacy (EuroSP), pp. 488–502. doi:10.1109/EuroSP.2018.00041.

[94] Radoglou-Grammatikis P.I., Sarigiannidis P.G., 2019. Securing the smart grid: A comprehensive compilation of intrusion detection and prevention systems. IEEE Access 7, pp. 46595–46620.

[95] Rahul Vigneswaran K., Vinayakumar R., Soman K., Poornachandran P., 2018. Evaluating Shallow and Deep Neural Networks for Network Intrusion Detection Systems in Cyber Security. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–6.

[96] Saganowski Ł., Goncerzewicz M., Andrysiak T., 2013. Anomaly Detection Preprocessor for SNORT IDS System. In: Advances in Intelligent Systems and Computing, pp. 225–232. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32384-3_28. URL https://doi.org/10.1007%2F978-3-642-32384-3_28.

[97] Sani Y., Mohamedou A., Ali K., Farjamfar A., Azman M., Shamsuddin S., 2009. An overview of neural networks use in anomaly Intrusion Detection Systems. In: 2009 IEEE Student Conference on Research and Development (SCOReD), pp. 89–92. doi:10.1109/SCORED.2009.5443289.

[98] Saxe J., Berlin K., 2015. Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), pp. 11–20. doi:10.1109/MALWARE.2015.7413680.

[99] Sewak M., Sahay S.K., Rathore H., 2018. Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection. In: 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 293–296. doi:10.1109/SNPD.2018.8441123.

[100] Shafahi A., Huang W.R., Najibi M., Suciu O., Studer C., Dumitras T., Goldstein T., 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In: Advances in Neural Information Processing Systems, pp. 6103–6113.

[101] Shang W., Cui J., Song C., Zhao J., Zeng P., 2018. Research on Industrial Control Anomaly Detection Based on FCM and SVM. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE. doi:10.1109/trustcom/bigdatase.2018.00042. URL https://doi.org/10.1109%2Ftrustcom%2Fbigdatase.2018.00042.

[102] Sharafaldin. I., Lashkari. A.H., Ghorbani. A.A., 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.

In: Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,, pp. 108–116. INSTICC, SciTePress. ISBN 978-989-758-282-0. doi:10.5220/0006639801080116.

[103] Shi Y., Sagduyu Y.E., 2017. Evasion and causative attacks with adversarial deep learning. In: MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM), pp. 243–248. ISSN 2155-7586. doi:10.1109/MILCOM.2017.8170807.

[104] Shi Y., Sagduyu Y.E., Davaslioglu K., Li J.H., 2019. Generative Adversarial Networks for Black-Box API Attacks with Limited Training Data. CoRR abs/1901.09113. URL http://arxiv.org/abs/1901.09113.

[105] Skansi S., 2018. Introduction to Deep Learning: from logical calculus to artificial intelligence. Springer.

[106] Skansi S., 2018. Recurrent Neural Networks, pp. 135–152. ISBN 978-3-319-73003-5. doi:10.1007/978-3-319-73004-2_7.

[107] Smolensky P., 1986. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science.

[108] Sonawane H.A., Pattewar T.M., 2015. A comparative performance evaluation of intrusion detection based on neural network and PCA. In: 2015 International Conference on Communications and Signal Processing (ICCSP), pp. 0841–0845. doi:10.1109/ICCSP.2015.7322612.

[109] Subba B., Biswas S., Karmakar S., 2016. A Neural Network based system for Intrusion Detection and attack classification. In: 2016 Twenty Second National Conference on Communication (NCC), pp. 1–6. doi:10.1109/NCC.2016.7561088.

[110] Szegedy C., Zaremba W., Sutskever I., Bruna J., Erhan D., Goodfellow I., Fergus R., 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 .

[111] Tavallaee M., Bagheri E., Lu W., Ghorbani A., 2009. A detailed analysis of the KDD CUP 99 data set. IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA 2. doi:10.1109/CISDA.2009.5356528.

[112] Tavallaee M., Bagheri E., Lu W., Ghorbani A.A., 2009. A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE symposium on computational intelligence for security and defense applications, pp. 1–6. IEEE.

[113] Tavoli R., et al. Providing a method to reduce the false alarm rate in network intrusion detection systems using the multilayer perceptron technique and backpropagation algorithm. In: 2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI), pp. 001–006. IEEE.

[114] Tomek I., 1976. Two Modifications of CNN. IEEE Transactions on Systems, Man, and Cybernetics SMC-6(11), pp. 769–772. ISSN 2168-2909. doi:10.1109/TSMC.1976.4309452.

[115] Totel E., Vivinis B., Mé L., 2004. A Language Driven Intrusion Detection System for Event and Alert Correlation. In: Proc. ot the 19th IFIP Int. Information Security Conf., pp. 209–224. Kluwer Academic.

[116] Ujjan R.M.A., Pervez Z., Dahal K., 2018. Suspicious Traffic Detection in SDN with Collaborative Techniques of Snort and Deep Neural Networks. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 915–920.

[117] Van N.T.T., Thinh T.N., 2015. Accelerating Anomaly-Based IDS Using Neural Network on GPU. In: 2015 International Conference on Advanced Computing and Applications (ACOMP), pp. 67–74. doi:10.1109/ACOMP.2015.30.

[118] Vigneswaran K.R., Vinayakumar R., Soman K., Poornachandran P., 2018. Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–6. IEEE.

[119] Vinayakumar R., Alazab M., Soman K.P., Poornachandran P., Al-Nemrat A., Venkatraman S., 2019. Deep Learning Approach for Intelligent Intrusion Detection System. IEEE Access 7, pp. 41525–41550. ISSN 2169-3536. doi:10.1109/ACCESS.2019.2895334.

[120] Vinayakumar R., Soman K.P., Poornachandran P., 2017. Applying convolutional neural network for network intrusion detection. In: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1222–1228.

[121] Wang E.K., Ye Y., Xu X., Yiu S.M., Hui L.C.K., Chow K.P., 2010. Security Issues and Challenges for Cyber Physical System. In: 2010 IEEE/ACM Int Conference on Green Computing and Communications & Int Conference on Cyber, Physical and Social Computing. IEEE. doi:10.1109/greencom-cpscom.2010.36. URL https://doi.org/10.1109%2Fgreencom-cpscom.2010.36.

[122] Wang P.H., Liao I.E., Kao K.F., Huang J.Y., 2018. An intrusion detection method based on log sequence clustering of honeypot for modbus TCP protocol. In: 2018 IEEE International Conference on Applied System Invention (ICASI), pp. 255–258. IEEE.

[123] Wang W., Sheng Y., Wang J., Zeng X., Ye X., Huang Y., Zhu M., 2018. HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection. IEEE Access 6, pp. 1792–1806. ISSN 2169-3536. doi:10.1109/ACCESS.2017.2780250.

[124] Wei Koh P., Steinhardt J., Liang P., 2018. Stronger Data Poisoning Attacks Break Data Sanitization Defenses.

[125] Xiao H., Biggio B., Brown G., Fumera G., Eckert C., Roli F., 2015. Is feature selection secure against training data poisoning? In: International Conference on Machine Learning, pp. 1689–1698.

[126] Xiao H., Biggio B., Brown G., Fumera G., Eckert C., Roli F., 2018. Is feature selection secure against training data poisoning? CoRR abs/1804.07933. URL http://arxiv.org/abs/1804.07933.

[127] Xu C., Shen J., Du X., Zhang F., 2018. An intrusion detection system using a deep neural network with gated recurrent units. IEEE Access 6, pp. 48697–48707.

[128] Xu C., Shen J., Du X., Zhang F., 2018. An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units. IEEE Access 6, pp. 48697–48707. ISSN 2169-3536. doi:10.1109/ACCESS.2018.2867564.

[129] Xu K., Li Y., Deng R.H., Chen K., 2018. DeepRefiner: Multi-layer Android Malware Detection System Applying Deep Neural Networks. In: 2018 IEEE European Symposium on Security and Privacy (EuroSP), pp. 473–487. doi:10.1109/EuroSP.2018.00040.

[130] Yang C., Wu Q., Li H., Chen Y., 2017. Generative poisoning attack method against neural networks. arXiv preprint arXiv:1703.01340 .

[131] Yeo M., Koo Y., Yoon Y., Hwang T., Ryu J., Song J., Park C., 2018. Flow-based malware detection using convolutional neural network. In: 2018 International Conference on Information Networking (ICOIN), pp. 910–913. doi:10.1109/ICOIN.2018.8343255.

[132] Yi Shi, Sagduyu Y., Grushin A., 2017. How to steal a machine learning classifier with deep learning. In: 2017 IEEE International Symposium on Technologies for Homeland Security (HST), pp. 1–5. doi:10.1109/THS.2017.7943475.

[133] Ying Wang, Yongjun Shen, Guidong Zhang, 2016. Research on Intrusion Detection Model using ensemble learning methods. In: 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp. 422–425. ISSN 2327-0594. doi:10.1109/ICSESS.2016.7883100.

[134] Yong L., Bo Z., 2019. An Intrusion Detection Model Based on Multi-scale CNN. In: 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), pp. 214–218. IEEE.

[135] Yoo H., Shon T., 2015. Novel approach for detecting network anomalies for substation automation based on IEC 61850. Multimedia Tools and Applications 74(1), pp. 303–318.

[136] Yuan X., 2017. PhD Forum: Deep Learning-Based Real-Time Malware Detection with Multi-Stage Analysis. In: 2017 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 1–2. doi:10.1109/SMARTCOMP.2017.7946997.

[137] Zhang J., Mani I., 2003. KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In: Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets.

Spis rysunków

Spis tablic

Zastosowanie Metod Uczenia Maszynowego do Wykrywania Ataków Sieciowych

Streszczenie

Każdego dnia liczba urządzeń i użytkowników korzystających z sieci zwiększa się. Wraz z wzrostem tej liczby, wzrasta też liczba wrogo nastawionych uczestników cyberprzestrzeni. Wśród tych urządzeń, poza komputerami osobistymi i telefonami, znajdują się też różne formy infrastruktury krytycznej (Critical Infrastructure – CI), a wkrótce dołączą do nich autonomiczne samochody. W takich okolicznościach zagadnienie cyberbezpieczeństwa staje się ważniejsze niż kiedykolwiek wcześniej. Trwający cybernetyczny wyścig zbrojeń powoduje ciągłe powstawanie nowych ataków, które wymagają bezustannej aktualizacji istniejących metod wykrywania tych ataków. Obecnie nawet same algorytmy uczenia maszynowego wykorzystywane do wykrywania ataków sieciowych stały się celem nowych wrogich działań. W związku z wyżej opisanymi problemami niniejsza praca ma dwa cele badawcze. Pierwszy – modyfikację znanych metod uczenia maszynowego dla lepszego wykrywania obecnych zagrożeń. W tym celu zbadano szereg możliwych ulepszeń schematu następujących po sobie procesów przetwarzania danych, łącznie z implementacjami najnowszych zdobyczy dziedziny uczenia maszynowego w domenie cyberbezpieczeństwa. Przebadano wpływ metod balansowania danych na nowoczesny zapis ruchu sieciowego, wpływ odpowiedniego doboru hiperparametrów na algorytmy oparte o sztuczne sieci neuronowe. Do działania w wykrywaniu ataków sieciowych przystosowano algorytm sieci Gated Recurrent Unit, oraz zaproponowano metodę opartą o ekstrakcję cech z użyciem Gated Recurrent Unit i klasyfikację przy pomocy algorytmu Random Forest. Drugi – opracowanie metody wykrywającej ataki typu evasion na algorytmy uczenia maszynowego wykorzystywane w cyberbezpieczeństwie. W tym celu zaimplementowano 4 różne ataki typu evasion, stworzono zbiór danych aktywacji neuronów sieci

wykorzystywanej do wykrywania ataków sieciowych, po czym przetestowano szereg metod uczenia maszynowego w roli klasyfikatora w metodzie wykrywania ataków typu evasion.

Słowa kluczowe: *Cyberbezpieczeństwo, uczenie maszynowe, sztuczna inteligencja*

The Application of Machine Learning Methods for Network Intrusion Detection

Abstract

With every single day the number of network users and connected devices increases. This expansion is followed by the inflating numbers of malicious cyberspace elements. Among the connected devices, besides personal computers and phones, one can find Internet of Things (IoT) devices, as well as various forms of Critical Infrastructure (CI), and autonomous vehicles are on their way to join that list. The current circumstances make cybersecurity an issue more pressing than ever before. The cybernetic arms race causes constant invention of new attacks, which in turn demands constant development of defensive and reactive measures. Currently, even the machine learning algorithms used for intrusion detection are under attack. Addressing that problem statement, the research objective of this work is two-fold. The first one: the modification of known machine learning methods for better intrusion detection in contemporary network traffic. To meet the objective, a variety of possible improvements to the data processing pipeline have been researched, along with the implementations of the newest achievements of machine learning in intrusion detection. The effect of data balancing methods on the contemporary network traffic is looked into, the influence of the correct selection of hyperparameters on the results achieved by machine learning methods is evaluated. The Gated Recurrent Unit algorithm is appropriated for intrusion detection, and a new method relying on the Gated Recurrent Unit for feature extraction and Random Forest for classification is proposed. The other objective – researching a method to detect adversarial evasion attacks on machine learning in intrusion detection. To meet that objective, 4 different evasion attacks are

implemented, a dataset of neural activations is collected, and a number of machine learning algorithms are tested for detection of evasion attacks.

Keywords: *Cybersecurity, Machine Learning, Artificial Intelligence*