# Wykorzystanie klasycznych metod uczenia maszynowego do wykrywania skanowania sieciowego z wykorzystaniem framework'a AB-TRAP

Jakub Szczepanowski
*Wydział Elektroniki i Technik Informacyjnych*
*Politechnika Warszawska*
Warszawa, Polska
01184204@pw.edu.pl

Alexander Załęski
*Wydział Elektroniki i Technik Informacyjnych*
*Politechnika Warszawska*
Warszawa, Polska
01142601@pw.edu.pl

***Streszczenie*—Celem pracy było wykorzystanie rozwiązań, które oferuje framework AB-TRAP w zadaniu wykrywania skanowania sieciowego. Wiązało się to z wybraniem infrastruktury ataku, zebraniu odpowiedniego zbioru danych, stworzeniem potoku uczenia maszynowego, wdrożeniem modelu do środowiska docelowego z użyciem wybranej metody oraz wykonaniem testów wydajnościowych. Sednem pracy było zapoznanie z konkretnym problemem systemów IDS w sektorze cyberbezpieczeństwa z zastosowaniem popularnych metod sztucznej inteligencji do jego rozwiązania.**

***Index Terms*—skanowanie sieciowe, IDS, sztuczna inteligencja, uczenie maszynowe**

## I. Wstęp

Rozwój sztucznej inteligencji znacząco przyczynił się do rozwoju wielu dziedzin związanych z analizą danych. Systemami wykorzystującymi analizę danych są systemy IDS należące do dziedziny cyberbezpieczeństwa. Służą one do wykrywania włamań do systemów komputerowych i dzielą się na dwa rodzaje: HIDS oraz NIDS. HIDS opiera swoje działanie na monitorowaniu działania aplikacji i systemu operacyjnego, natomiast NIDS na analizie ruchu sieciowego. Wykrywanie podejrzanych aktywności na poziomie sieci polega między innymi na detekcji skanowania portów. Zespół brazylijskich naukowców opracował rozwiązanie dla systemu NIDS wykorzystującego techniki uczenia maszynowego stricte pod to zagadnienie - framework AB-TRAP. Modelowanie danych, szczególnie w formie uczenia wsadowego, swoje dobre działanie opiera na wykorzystaniu możliwie najnowszego zbioru danych odzwierciedlającego aktualne techniki ataku na sieć stosowane przez cyberprzestępców. AB-TRAP zawiera procedury trenowania na najnowszych dostępnych danych oraz wdrożenia modeli uczenia maszynowego wykorzystując przy tym zarówno środowisko lokalne (sieć LAN) jak i globalne (sieć Internet). Użycie framework'a bazuje na kilku krokach: wygenerowanie sztucznego zbioru pakietów symulującego atak na sieć, wygenerowanie zbioru pakietów reprezentujących legalny ruch sieciowy, wstępne przetwarzanie i trenowanie modeli, wdrożenie modeli do urządzenia docelowego oraz jądra systemu operacyjnego w środowisku użytkownika, ocenę wydajności modeli na urządzeniu docelowym. Poniższa praca stanowi zastosowanie tych kroków w praktyce ograniczając się do przypadku sieci lokalnej oraz wdrożenia opartego o moduł jądra linuksowego jako przetestowania skuteczności działania framework'a w zadaniu wykrywania skanowania sieciowego.

## II. Zbiór danych

Każde zadanie uczenia maszynowego opiera się na zbiorze danych. W przypadku zadania, jakim jest wykrywanie skanowania sieciowego, będzie on w sposób naturalny zawierał informacje na temat ruchu sieciowego. Istotną kwestią jest typ i format tego ruchu. Z pewnością konieczne są dane pakietów związanych z atakiem. Powinny być one wystarczająco zróżnicowane. Najlepiej, jakby reprezentowały wiele różnych technik skanowania pochodzących z różnych programów wykorzystywanych przez cyberprzestępców. Oprócz przykładów pozytywnych(związanych z atakiem), w zbiorze muszą znajdować się również przykłady negatywne(zwykły ruch sieciowy). Należy zachować między nimi odpowiednie proporcje, tak aby zapewnić jak najlepszy stopień generalizacji modelu. Jeżeli system IDS tworzony jest dla konkretnego przedsiębiorstwa/sieci to naturalnym byłoby przechwycenie jak największej ilości pakietów stanowiących typowy/zwyczajny dla takiego przypadku ruch. Jeżeli projekt nie jest specjalizowany pod konkretnego klienta należy posiłkować się publicznymi bazami danych jak najlepiej odzwierciedlającymi typowy, legalny ruch sieciowy. Jedną z takich baz jest baza MAWILab. Stanowi ona ogromny zbiór pakietów przepływających przez sieć i jest aktualizowana codziennie co gwarantuje poprawne odzwierciedlenie obecnych trendów. Format samych pakietów zależy od zastosowanego sniffera. Konwersja do struktury tabelarycznej jest wymagana do realizacji zadania uczenia maszynowego. Framework AB-TRAP dostarcza oba gotowe zbiory: zbiór ataku - pozyskany poprzez symulację ataku w infrastrukturze zarówno lokalnej jak i internetowej, zbiór

legalnego ruchu - powstały w wyniku przekształceń zbioru MAWILab tak, aby uzyskać jak najlepsze wyniki uczenia. Zbiór ataku powstał poprzez samo przechwytywanie pakietów i przekonwertowanie ich, natomiast zbiór legalnego ruchu wymagał dodatkowych przekształceń. Skoro zbiór ten ma reprezentować jak najmniej „podejrzany" ruch należy się z niego pozbyć danych odstających. Ponadto każde przykłady znacząco różniące się od reszty źle wpływają na uogólnienie całości. Dane dotyczące anomalii są rejestrowane i zapisywane do pliku CSV przez MAWILab. Na podstawie tych danych twórcy AB-TRAP'a wyekstrahowali filtr i przefiltrowali przez niego pakiety. Następnie dokonano losowego próbkowania zmniejszając jednocześnie wielkość całości, którą pod koniec całego procesu przekonwertowano do formatu CSV.

## III. Wykorzystane modele

W poniższej sekcji zostały opisane algorytmy wykorzystane do stworzenia modeli do celów wykrywania skanowania.

W niniejszej pracy zostały wykorzystane poniższe modele:

- Naiwny klasyfikator Bayesa - opiera się na prawdopodobieństwie przyjmowania danej wartości cechy opisywanej pod warunkiem określonych wartości cech opisujących,
- Regresja logistyczna - metoda „opakowania" modelu liniowego w funkcję logistyczną do wyznaczania predykcji prawdopodobieństwa klasy 1,
- Drzewo decyzyjne - dzieli przestrzeń cech opisujących na przedziały, w których cecha opisywana przyjmuje daną wartość. W przypadku klasyfikacji binarnej z niniejszej pracy są to wartości 0 lub 1,
- Las losowy - metoda zespołowa złożona z wielu słabszych modeli bazowych jakimi są drzewa decyzyjne z wykorzystaniem randomizacji cech oraz prób bootstrapowych,
- Maszyna wektorów nośnych (SVM) - metoda opierająca się na separowalności liniowej przykładów w przestrzeni cech z uzyskaniem jak największego marginesu,
- K-najbliższych sąsiadów - klasteryzacja na podstawie wybranej liczby sąsiadów przewidywanego przykładu.

## IV. Wykorzystane metryki

W przypadku klasyfikacji możliwe jest zastosowanie następujących metryk jakości modelu:

- Dokładność (ang. accuracy) - liczba poprawnie sklasyfikowanych próbek podzielona przez całkowitą liczbę próbek. Można ją stosować w przypadku zbilansowanych klas, jednak w przypadku klas niezbilansowanych miara ta może wprowadzać w błąd.
- Precyzja (ang. precision) - liczba poprawnie sklasyfikowanych próbek o wartości pozytywnej podzielona przez liczbę wszystkich próbek sklasyfikowanych przez model jako pozytywne.
- Zupełność (ang. recall) - liczba poprawnie sklasyfikowanych próbek o wartości pozytywnej podzielona przez liczbę wszystkich próbek o wartości pozytywnej.
- Miara f1 (ang. f1 score) - miara łącząca precyzję i zupełność równa: f1 = 2 * (precyzja * zupełność) / (precyzja + zupełność). W przypadku klas niezbalansowanych dobrze oddaje jakość modelu.

## V. Przygotowanie danych i trenowanie modeli

Cały proces tworzenia modeli uczenia maszynowego zaczyna się od zbioru oraz jego wstępnego przetwarzania. Wykorzystany zbiór danych powstał poprzez konkatenację dwóch zbiorów opisanych w części poświęconej temu zagadnieniu: zbioru ataku oraz legalnego ruchu. Było również wspomniane jak ważne jest zastosowanie odpowiednich proporcji między nimi. Te są już zachowane w składowych zbiorach i są następujące: zbiór ataku - 86 480, zbiór legalnego ruchu - 135 558, co daje stosunek około 1,5:1 na rzecz legalnego ruchu i jest to jak najbardziej logiczne przy założeniu, że większość obserwowanego ruchu to ruch legalny. Jednocześnie próbki ataku nie stanowią w nim absolutnej mniejszości tak, aby całość nie stała się niezbalansowana. Pierwszym etapem przetwarzania wstępnego było dodanie do każdej pobranej ramki etykiet binarnych oznaczających atak lub brak ataku. Przez to, że całe skanowanie opiera się o protokół transportowy TCP wycięto rekordy niepasujące do tego założenia (dokładnie 16 397). Następnie przeprowadzono analizę cech, w tym analizę typów danych, wariancji oraz pól brakujących. Redukcja wymiarowości w tym konkretnym zadaniu opiera się głównie na pozbyciu się atrybutów, które są nadmiarowe: albo nie wnoszą żadnej informacji albo są nieadekwatne do rozwiązania problemu. Bardzo wybrakowane kolumny: 'ip.tos' i 'tcp.options.mss_val' automatycznie zostały wyrzucone ze zbioru, aby nie narażać się na błędy związane ze sztucznym ich wypełnianiem, szczególnie, że braki te stanowiły niemalże cały zbiór. Atrybuty odnoszące się do warstwy łącza danych również były do wyrzucenia: 'frame_info.time', 'frame_info.encap_type', 'frame_info.time_epoch', 'frame_info.number', 'frame_info.len', 'frame_info.cap_len', 'eth.type'. Adresy IP zostały usunięte, żeby nie uzależniać modelu od konkretnych przypadków. Informacje o wersjach protokołów są zbyteczne, a zsumowane flagi zarówno TCP jak i IP są redundantne, ponieważ zbiór dysponuje ich składowymi. Usunięto również atrybuty o zerowej wariancji: 'ip.hdr_len', 'ip.flags.rb', 'ip.flags.mf', 'ip.frag_offset'. Ostatnie usunięte atrybuty to sumy kontrolne ('ip.checksum', 'tcp.checksum'), numery sekwencyjne (tcp.seq', 'tcp.ack') oraz numer TTL ('ip.ttl'). Po redukcji wymiarowości zostało 18 cech w tym dwie posiadające wartości heksadecymalne, które zostały przekonwertowane do systemu dziesiętnego. Wykaz histogramów w podziale na pakiety ataku i legalnego ruchu to głównie dysproporcje między przesyłanymi flagami. Najbardziej złożone rozkłady to źródłowe i docelowe porty. Jest widoczny znaczący spadek jakości predykcji po usunięciu tych atrybutów, co świadczy o ich istotności. Oczywiście, porty często mogą być losowe co nie oznacza, że nie ma w nich żadnych prawidłowości. Konkretne usługi i programy używają często tych samych numerów, albo należących do konkretnego przedziału. Modalność w obu przypadkach jest podobna, ale różnią się one skalą przez co są między sobą

rozróżnialne. Kolejnym etapem było usunięcie rekordów z brakującymi danymi (były dwa takie rekordy, więc nie wpłynęło to w żaden sposób na ostateczne wyniki) i podzielenie zbioru na cechy opisujące i cechę opisywaną. Ostatnie procedury przetwarzania wstępnego to podział na dane treningowe i testowe - zdecydowano się na podział w stosunku 2:1 oraz standaryzację danych - skalowanie cech polegające na odjęciu wartości średniej a następnie podzieleniu przez odchylenie standardowe. Tak przygotowane dane były gotowe do zastosowania w wybranych modelach. Uczenie z wykorzystaniem wymienionych wcześniej algorytmów zrealizowano na podstawie przeszukiwania siatki. Każda regularyzacja hiperparametryczna używała walidacji krzyżowej z podziałem na 5 podzbiorów, natomiast ocena była dokonywana z użyciem metryki f1. Dla każdego algorytmu należało sporządzić słownik hiperparametrów sprawdzanych w procesie uczenia. Na koniec dla najlepszego zestawu hiperparametrów modele zostały wytrenowane na całych danych treningowych i została określona jakość predykcji. W przypadku naiwnego klasyfikatora bayesowskiego było to tylko wygładzanie, które nie wpływa drastycznie na jakość wyników, dlatego słabość tego rozwiązania zależy głównie od charakterystyki algorytmu. Regresja logistyczna posiada ich już więcej i są to: maksymalna liczba iteracji, kara - lasso (L1), ridge (L2), współczynnik intensywności kary oraz optymalizator. Słabością tego rozwiązania może być silna korelacja między zmiennymi niezależnymi. W wykorzystywanym zbiorze występują zależności liniowe np. między poszczególnymi flagami (ACK i SYN, ACK i DF). W przypadku drzewa decyzyjnego rozważano kryteria wyboru (współczynnik giniego, entropia), maksymalną głębokość drzewa, minimalną liczbę przykładów do dokonania podziału, minimalną liczbę przykładów do ustanowienia liścia, maksymalną liczbę atrybutów zastosowanych do budowy drzewa. Algorytm drzewa decyzyjnego nie jest wrażliwy na korelację cech ani ich liczbę co jest dobrą prognozą dla jego efektywności w tym konkretnym zadaniu. Hiperparametry rozważane dla maszyny wektorów nośnych to współczynnik intensywności kary, maksymalna liczba iteracji, funkcja jądrowa (liniowa, radialna, wielomianowa), stopień (tylko dla jądra wielomianowego). Duża liczba cech oraz przykładów może być dla SVM dużym obciążeniem. Niuanse decydujące o tym, czy dany pakiet będzie sklasyfikowany jako atak lub legalny ruch jest dodatkowym utrudnieniem w kontekście separacji liniowej, którą wykonuje. K-najbliższych sąsiadów okazał się być w tym temacie bardziej obiecujący. W projekcie rozważano liczbę najbliższych sąsiadów tylko dla odległości euklidesowej. Zastosowanie grupowania jest intuicyjnym podejściem w tym przypadku. Ostatnim zastosowanym algorytmem był las losowy. Przeszukiwano takie same hiperparametry jak w przypadku jednego drzewa decyzyjnego plus ilość estymatorów w zespole. Algorytm ten jest tak samo obiecujący jak pojedyncze drzewo, a nawet bardziej gdyż korzysta z wielu modeli bazowych co daje większe szanse na uzyskanie dobrego wyniku.

## VI. Najlepszy model

Najlepszy model został określony na podstawie wartości miary f1 wyznaczonej na zbiorze testowym. W poniższej tabeli zostały przedstawione wyniki miary f1 dla przetestowanych modeli:

| model | f1_score |
|---|---|
| random_forest_classifier | 0.997307 |
| decision_tree_classifier | 0.996195 |
| k_neighbors_classifier | 0.981305 |
| logistic_regression | 0.818610 |
| gaussian_nb | 0.646485 |
| svc | 0.602050 |

Na podstawie powyższych wyników najlepszym modelem jest las losowy z wynikiem f1 równym 0.997307. Wyniki zauważalnie lepsze należą do trzech pierwszych algorytmów. Potwierdzają one uzasadnienia o ich skuteczności przytoczone w poprzednim punkcie. Ze względu na małą różnicę w ocenie między lasem a pojedynczym drzewem, zdecydowano się na użycie drzewa jako modelu, ponieważ zwiększy to wielokrotnie wydajność całego systemu nie zmniejszając przy tym znacznie jakości predykcji. Filtry sieciowe wykonując wymagającą analizę mogą nie nadążać za ruchem sieciowym i aby go nie spowalniać mogą przepuszczać część pakietów co nie powinno mieć miejsca w tego typu rozwiązaniu.

## VII. Wdrożenie

Systemy IDS muszą nieustannie analizować ruch sieciowy, przez co są dużym obciążeniem dla sprzętu. Z tego względu wskazana jest konwersja uzyskanego rozwiązania do postaci niskopoziomowej. Funkcję tę realizuje moduł emlearn służący do przekształcenia modeli uczenia maszynowego do kodu języka C w standardzie C99 [1]. Jak na dzisiejsze czasy jest to stary standard, co pozwala na wdrożenie opisywanego rozwiązania na dużej liczbie urządzeń. Twórcy jako urządzenia docelowe wskazują mikrokontrolery i systemy wbudowane, czyli urządzenia z minimalnymi zasobami. Świadczy to o jakości i stopniu optymalizacji samego rozwiązania. Dla przykładu kod w C reprezentujący drzewo decyzyjne będzie masywnym łańcuchem zagnieżdżonych instrukcji warunkowych. Taki kod można zawrzeć w LKM (Loadable Kernel Module), czyli module jądra linuksowego. „Poruszając się" w obszarze jądra systemu operacyjnego potrzeba specjalnych mechanizmów, które będą stanowiły interfejs między dopiero co pojawiającym się pakietem a dalszym jego przetwarzaniem co umożliwi jego analizę. Preferowanym przez AB-TRAP rozwiązaniem w tej dziedzinie jest framework Netfilter. Udostępnia on zestaw funkcji przyjmujących wywołania zwrotne, które zostają użyte w momencie wskazanym przez programistę np. tak jak w tym przypadku przed procesem rutingu (NF_INET_PRE_ROUTING). Parametrem filtra jest bufor, z którego można wyekstrahować struktury jądra linuksa składające się na implementację protokołu między innymi TCP i IP. Konstrukcja przekonwertowanego modelu jest uniwersalna,

nie odnosi się ona do nazw kolumn, tylko do indeksów atrybutów przekazanych w postaci tablicy. Pierwszą zatem rzeczą, którą należało zmienić było odwołanie nie poprzez indeks a przez nazwę pola struktury. Kolejna rzecz to liczby, do których są porównywane pola. Powinny być one w formacie sieciowym jakim jest big-endian (najbardziej znaczący bajt znajduje się na początku) z wyjątkiem jednego atrybutu, który w procedurze jego obliczania został przepuszczony przez funkcję ntons. Funkcja, która realizuje konwersję przyjmuje dwubajtową liczbę całkowitą jako argument. Dane w strukturze modelu są zestandaryzowane przez co po rzutowaniu na typ całkowitoliczbowy straciły by całą informację. Na szczęście algorytmy związane z drzewami decyzyjnymi nie są wrażliwe na skalowanie cech, przez co możliwe jest ponowne wytrenowanie modelu na pierwotnych/niezmienionych wartościach. Wyszło to nawet ostatecznie na dobre dla całego rozwiązania, ponieważ otrzymano nieco lepszy wynik. Ostatnia istotna zmiana to zmiana relacji z nierówności na równość wszędzie tam, gdzie porównanie było dokonywane na flagach (posiadają tylko dwie wartości - 0 lub 1). Tak przekształcony model wklejono do przygotowanego pliku napisanego w języku C przystosowanego do rejestracji i usunięcia filtra. Funkcja filtra po otrzymaniu predykcji weryfikuje ją - jeżeli otrzymano klasę pozytywną logowany jest komunikat o wykryciu skanowania sieciowego i pakiet jest odrzucany, poprzez zwrócenie wartości NF_DROP, w przeciwnym wypadku pakiet jest kierowany dalej, poprzez zwrócenie NF_ACCEPT. Plik ten został następnie skompilowany za pomocą programu *make* w celu otrzymania wynikowego pliku z rozszerzeniem *.ko*. Z wykorzystaniem komendy *insmod* moduł jądra został do niego załadowany i uruchomiony.

## VIII. Testy funkcjonalności

Do testów używano wirtualnej infrastruktury opartej o sieć NAT. Maszynie „cel" zostały przydzielone cztery rdzenie procesora oraz 4GB pamięci RAM. Stworzono także maszynę „atakujący". Sprawdzono przy tym trzy różne moduły oparte o: las losowy z uwzględnieniem portów, drzewo decyzyjne bez i z uwzględnieniem portów, z których wybrano drzewo z uwzględnieniem portów. Testy funkcjonalności sporządzono w oparciu o skrypt wykorzystujący dwa programy do skanowania: nmap oraz hping3. Podczas testów użyto różnych metod skanowania: TCP SYN Scan, TCP Connect Scan, TCP NULL Scan, TCP XMAS Scan, TCP FIN Scan, TCP ACK Scan, TCP Window Scan oraz TCP Maimon Scan. W czasie wykonywania skryptu system momentalnie wyrzucał do logów jądra komunikaty o wykryciu skanowania sieciowego dla każdej z powyższych metod. W stanie spoczynku nie wykazywał żadnych niepożądanych zachowań.

## IX. Testy wydajności

Wydajność, w tym stopień obciążenia modułu wywieranego na sprzęt komputerowy zbadano z wykorzystaniem programu IPerf3 służącego do pomiaru przepustowości sieci, poprzez generowanie pakietów TCP i UDP między dwoma hostami. Do zbierania i zapisywania statystyk systemowych użyto narzędzia sar (System Activity Reporter), a do konwersji do formatu CSV użyto narzędzia sadf (System Activity Data Collector), oba należą do pakietu systemowego sysstat. Program IPerf3 generuje kilkadziesiąt megabajtów na sekundę co wystawia system wykrywania na próbę. Po zgromadzeniu 1200 pomiarów można było porównać wyniki. Oba pomiary: z działającym modułem i bez niego są niemalże identyczne. Średnie zużycie procesora oscyluje w okolicach 6%, natomiast zużycie pamięci to około 4%.

## X. Podsumowanie

Cel pracy został zrealizowany z nawiązką poprzez usprawnienie dotyczące uwzględnienia portów w procesie uczenia modelu. Wykonano wszystkie kroki oferowane przez framework co z pewnością usprawniło cały proces tworzenia projektu. Analizując wyniki końcowe: referencyjnym algorytmem do sporządzania rozwiązania problemu skanowania sieciowego jest drzewo decyzyjne. W porównaniu z lasem losowym, który uzyskał nieco lepszy wynik oceny jakości predykcji uzyskuje on wielokrotnie większą wydajność. Las losowy nie nadążał za ruchem sieciowym przez co jego predykcje były zdecydowanie rzadsze, niż w przypadku pojedynczego drzewa. Drzewo bez uwzględnienia portów tak, jak było w pierwotnym zaleceniu AB-TRAP'a, nie sprawdziło się tak dobrze, ponieważ oprócz poprawnej predykcji podczas skanowania średnio co kilka sekund model wyrzucał wykrycie skanowania w stanie spoczynku co oznaczało, że interpretował w pewnym stopniu zwyczajny ruch jako podejrzany. Obciążenie modelu na działanie całego systemu komputerowego było niezauważalne. Framework AB-TRAP jest przydatnym narzędziem do budowy systemu IDS w kontekście wykrywania skanowania sieciowego.

## References

[1] Jon Nordby. emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices. mar 2019, 10.5281/zenodo.2589394, https://doi.org/10.5281/zenodo.2589394

## Literatura

[1] Gustavo De Carvalho Bertoli, LourençoAlves Pereira Júnior, Osamu Saotome, Aldri LDos Santos, Filipe Alves Neto Verri, CesarAugusto Cavalheiro Marcondes, Sidnei Barbi-eri, Moises S Rodrigues, and José M ParenteDe Oliveira. An end-to-end framework for machinelearning-based network intrusion detection system.IEEE Access, 9:106790–106805, 2021.