Jakub BOTWICZ
POLITECHNIKA WARSZAWSKA, INSTYTUT TELEKOMUNIKACJI

Implementacja procesu klasyfikacji danych z użyciem układów reprogramowalnych

Mgr inż. Jakub BOTWICZ

Absolwent studiów magisterskich na wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Obecnie uczestnik studiów doktoranckich w Instytucie Telekomunikacji tej uczelni. Główne obszary pracy naukowej to: bezpieczeństwo systemów teleinformatycznych oraz wydajna i efektywna implementacja algorytmów w układach rekonfigurowalnych.

e-mail: J.Botwicz@elka.pw.edu.pl

Streszczenie

W artykule opisano różne problemy klasyfikacji danych oraz podano dziedziny w których mają one zastosowanie. Następnie przedstawiono architekturę systemu, w którym będzie możliwe zaimplementowanie podanych wcześniej przez innych autorów, sprawdzonych już algorytmów klasyfikacji danych i wsparcie ich działania poprzez specjalizowane układy sprzętowe. Podano wyniki (w postaci skuteczności klasyfikacji oraz zużycia zasobów) przykładowych modułów sprzętowych. Przedstawiony został również proces tworzenia modułu sprzętowego – od danych wejściowych poprzez wygenerowany kod źródłowy w języku opisu sprzętu, aż po konfigurację układu reprogramowalnego.

Słowa kluczowe: klasyfikacja danych, akceleracja obliczeń za pomocą architektur sprzętowych, bezpieczeństwo systemów teleinformatycznych, wyszukiwanie wzorców.

Implementation of data classification process using reconfigurable hardware

Abstract

In this article various classification problems was described and also their applications was depicted. Afterwards the hardware module architecture was introduced in which there is a possibility to implement previously described mature classification algorithms. The article contains results of testing hardware classification modules (classification precision and hardware resources usage). Finally, the complete process of module generation was presented (from examples of data, through source code in hardware description language to reconfigurable hardware configuration).

Keywords: data classification, computing acceleration using hardware architectures, IT security, pattern matching.

1. Wstęp

Sieć *Internet* jest w chwili obecnej powszechnym medium wymiany informacji i stanowi prawdziwy układ nerwowy światowej gospodarki oraz społeczeństwa. Ilość danych przesyłana i udostępniana w tej sieci a także w sieciach wydzielonych wymaga stworzenia narzędzi zabezpieczających i wspomagających ich przetwarzanie.

Podstawową metodą na zwiększenie bezpieczeństwa systemów teleinformatycznych jest podział całego systemu na rozdzielne segmenty i kontrolowanie danych wpływających z zewnątrz systemu oraz przepływających pomiędzy poszczególnymi segmentami za pomocą zapór przeciwogniowych (*firewalls*), systemów przeciwdziałania włamaniom (*intrusion prevention systems*), bram skanujących pocztę elektroniczną (*anti-virus mail gateways*), itp. Dla analizy danych przekazywanych za pomocą poczty elektronicznej i protokołów wymiany plików niezbędne jest określenie, jakie dokumenty zawierają istotne informacje, w jaki sposób można je kategoryzować oraz jak wykrywać dane potencjalnie niebezpieczne. Proces ten jest bardzo wymagający obliczeniowo i powszechnie używane obecnie procesory ogólnego zastosowania są coraz częściej wąskim gardłem w szybkim przetwarzaniu danych [5].

Pomimo tego, że procesory ogólnego zastosowania ciągle zwiększają swoje możliwości, ich maksymalną wydajność ogranicza prawo Moore'a. Również technologie stosowane do przesyłania danych w sieci Internet mają podobne ograniczenia, jednak standardowe architektury procesorowe rozwijają się wolniej, co wymusza potrzebę rozwoju specjalizowanych architektur sprzętowych — takich jak układy *Application Specific Integrated Circuits* (*ASIC*) oraz *Field Programmable Gate Arrays* (*FPGA*). Obie architektury pozwalają na zrównoleglenie i spotokowanie pracy układu, dzięki czemu można pozbyć się ograniczeń przeszkadzających w rozwoju procesorów ogólnego zastosowania. Szczególnie architektura układów *FPGA* jest bardzo użyteczna dla tworzenia i prototypowania nowych rozwiązań z użyciem sprzętowej akceleracji obliczeń.

Celem tego artykułu jest pokazanie architektury systemu, w którym będzie możliwe zaimplementowanie istniejących, sprawdzonych już algorytmów klasyfikacji danych i wsparcie ich działania poprzez specjalizowane układy sprzętowe.

2. Klasyfikacja plików na podstawie ich zawartości

W chwili obecnej większość firm i organizacji daje swoim pracownikom możliwość dostępu do sieci *Internet*. Pozwala to wysyłanie oraz odbieranie plików za pośrednictwem poczty elektronicznej, przeglądarek internetowych lub połączeń bezpośrednich. W organizacjach dbających o bezpieczeństwo przetwarzanych informacji ten proces jest zwykle obwarowany *polityką bezpieczeństwa*, zawierającą reguły określające, jakie rodzaje dokumentów można przesyłać do i z sieci zewnętrznej. Aby skutecznie wdrażać taką politykę niezbędne są programy potrafiące automatycznie i bez udziału operatora-nadzorcy rozpoznawać typy plików na podstawie ich zawartości (*content-based filetypes classification*). Poza skutecznością klasyfikacji, dodatkowym problemem jest potrzeba zapewnienia dużej przepustowości przetwarzania danych przez tego typu programy.

Zwyczajowo każdy rodzaj pliku ma swoje określone rozszerzenie (czyli ostatnią część nazwy po kropce). Dodatkowo niektóre pliki mają w swoich definicjach określoną wartość w nagłówku - np. pliki *Portable Document Format* rozpoczynają się znakami %PDF. Niestety ta metoda identyfikacji może być łatwo oszukana i z tego powodu nie jest skuteczna. Najlepszym rozwiązaniem tego problemu byłoby powszechne zastosowanie infrastruktury klucza publicznego poprzez tworzenie podpisu elektronicznego dla każdego z przesyłanych plików. Niestety w chwili obecnej ta idea dopiero zyskuje na popularności i w związku z tym konieczne są rozwiązania łatwiejsze do wdrożenia.

Analiza i pełne parsowanie każdego typu plików wymagałyby znajomości specyfikacji wszystkich standardów oraz ciągłego ich uaktualniania w kontakcie z producentami oprogramowania je wytwarzającymi. To zadanie jest niewykonalne z powodu olbrzymiej liczby różnorodnych typów plików oraz braku dostępu do specyfikacji niektórych formatów (np. pliki tworzone przez pakiet *Microsoft Office*).

2.1. Przegląd zaproponowanych rozwiązań

Rozwiązaniem nie wymagającym pełnego parsowania zawartości plików jest użycie metod z dziedziny sztucznej inteligencji (artificial intelligence) – takich jak uczenie maszynowe (machine learning) [2]. Problem ten można

sformułować jako n-klasowa kategoryzacja gdzie liczba n odpowiada liczbie typów plików jakie potrzebujemy rozróżniać. Do tak postawionego problemu najlepiej pasuje schemat uczenia z nadzorem (supervised learning) ponieważ możemy przygotować zbiór wstępnie sklasyfikowanych przykładów, które będą służyć do uczenia klasyfikacji i oceny rezultatów tego procesu. Uczenie klasyfikacji danych będzie się sprowadzało do wykonania następujących kroków:

1. zebranie zbioru przykładów i ich klasyfikacja na ustalone kategorie
2. ekstrakcja i wybór cech na podstawie których będzie dokonywana klasyfikacja
3. przygotowanie klasyfikatora o ustalonych parametrach takich jak dokładność klasyfikacji czy też czas działania

Najbardziej istotną i jednocześnie najtrudniejszą fazą tego procesu jest ekstrakcja i wybór cech. Od powodzenia tego etapu zależy skuteczność całego procesu uczenia klasyfikacji. Zbiór cech może nie zawierać informacji niezbędnych do poprawnej klasyfikacji. Z drugiej strony, jeśli cech będzie zbyt dużo, to proces przygotowania klasyfikatora będzie bardzo kosztowny obliczeniowo. Najczęściej stosowanymi metodami analizy danych tekstowych czy też binarnych są:

- **jedno-znakowa**: analiza występowania poszczególnych znaków (zazwyczaj traktowanych jako jeden bajt, czasem jako dwa bajty w kodzie *Unicode*),
- **n-gramowa**: analiza występowania ciągu bajtów o długości n [3],
- **słownikowa**: analiza występowania słów ze słownika stworzonego w procesie uczenia.

Wszystkie wymienione wyżej metody mogą dawać atrybuty binarne (czyli odpowiadać na pytanie czy znak, n-gram lub słowo występują w przetwarzanym fragmencie tekstu) lub wielowartościowe (czyli reprezentować częstość występowania znaku, n-gramu lub słowa), jednak zazwyczaj metoda jedno-znakowa i słownikowa są związane z atrybutami wielowartościowymi, natomiast analiza n-gram z atrybutami binarnymi.

3. Wykrywanie złośliwego oprogramowania

Termin *złośliwe oprogramowanie* (*malicious code*) jest definiowany bardzo ogólnie jako wszelki kod wykonywalny, który został stworzony w celu zadania szkód użytkownikom systemów teleinformatycznych [1]. W tej kategorii mieszczą się: wirusy, robaki, konie trojańskie oraz wiele innych (w literaturze bardzo często złośliwe oprogramowanie nazywane jest po prostu wirusami i również w tym artykule, ze względu na prostotę językową, będę używał takiego uproszczenia). Automatyczne wykrywanie złośliwego oprogramowania (*malicious code identification*) jest istotne ze względu na duże straty, jakie może spowodować skasowanie, kradzież danych lub zablokowanie dostępu do usług, które może być dokonane za pomocą tego typu programów. Stosowanie programów antywirusowych jest obecnie powszechne, jednak ich działanie jest zazwyczaj oparte na sygnaturach znanych wirusów. Takie rozwiązanie pozwala uzyskać dużą wydajność tych systemów oraz precyzję działania, jednak wymaga ciągłego uzupełniania bazy sygnatur oraz szybkiej dystrybucji do użytkowników.

Zadanie wykrywania złośliwego oprogramowania jest powiązane z zadaniem klasyfikacji typu pliku według zawartości, ponieważ klasyfikacja formatu pliku powinna być wykonywana przed określaniem, czy oprogramowanie jest złośliwe i bardzo często wymaga odpowiedzi na pytania: czy jest to skrypt czy plik wykonywalny; w przypadku skryptów: w jakim języku został napisany skrypt; w przypadku plików wykonywalnych: dla jakiego systemu operacyjnego oraz dla jakiej platformy procesorowej został przygotowany ten plik. Jeśli klasyfikacja typu pliku zostanie wykonana nieprawidłowo, to mogą wystąpić błędy identyfikacji złośliwego oprogramowania: na przykład dokument tekstowy opisujący wirusy i zawierający z tego powodu ich sygnatury może zostać rozpoznany jako wirus.

Problem identyfikacji złośliwego oprogramowania jest typowym problemem 2-klasowej kategoryzacji. Jednak w porównaniu do wcześniejszego problemu, specyfika plików wykonywalnych pozwala nam na użycie odmiennych metod ekstrakcji cech. W artykule [6] zaprezentowano różne metody wstępnego przetwarzania pliku dla potrzeb ekstrakcji cech a następnie przedyskutowano ich wady i zalety. Metody te to:

- wyodrębnienie z pliku łańcuchów tekstowych,
- profilowanie wywołań funkcji,
- pełen zrzut zawartości pliku w postaci heksadecymalnej.

Pierwsza metoda polega na wyodrębnieniu z sekcji *data* lub *sdata* badanego pliku wykonywalnego wszystkich drukowalnych łańcuchów tekstowych. Sposób ten jest bardzo łatwy do manipulacji, ze względu na łatwość umieszczenia w kodzie wirusa dokładnie takich łańcuchów tekstowych, jak w innym poprawnym programie.

Druga metoda polega na wyodrębnieniu z każdego pliku listy bibliotek dołączanych dynamicznie (*dynamically linked libraries*) oraz utworzenia dla badanego pliku rozkładu wywołań do poszczególnych funkcji. Można w ten sposób określić czy program korzysta z konsoli tekstowej, bibliotek graficznych, dostępu do sieci czy też różnych urządzeń zewnętrznych. Niestety wymaga to dosyć skomplikowanej analizy (parsowania) i w związku z tym jest bardzo czasochłonne obliczeniowo.

Autorzy artykułu [7] korzystali ze zbioru danych składającego się z 1971 prawidłowych programów oraz 1651 przypadków złośliwego oprogramowania. Wszystkie programy były plikami wykonywalnymi dla systemu *Windows*. Do ekstrakcji cech autorzy wykorzystali analizę n-gramów. Przetwarzając zbiór trenujący przygotowano 69 milionów unikalnych n-gramów. Następnie używając współczynnika przyrostu informacji (*informational gain*) wybrano 500 najlepszych n-gramów o długości 4 bajtów. Proces ekstrakcji cech (czyli wykrywania wzorców wybranych tetragramów) wymagał najwięcej mocy obliczeniowej z całego procesu klasyfikacji plików. Na tak stworzonym zbiorze przykładów autorzy przetestowali różne metody tworzenia klasyfikatorów i najlepsze wyniki uzyskali dla drzew decyzyjnych. Ostatecznie uzyskano błąd klasyfikacji mniejszy niż 0,4%.

Autorzy stwierdzili, że skomplikowane metody (profilowanie wywołań funkcji oraz wyodrębnienie łańcuchów tekstowych) były wysoce nieefektywne, a jednocześnie dawały wyniki gorsze lub takie same jak metoda najprostsza czyli zrzut danych w postaci heksadecymalnej i wykonanie analizy n-gram.

4. Proponowane rozwiązanie

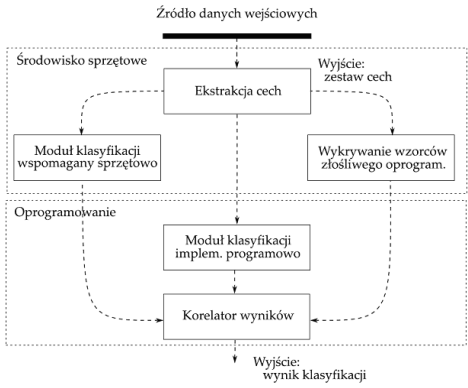
Przeanalizowane artykuły jasno wskazują, że najlepsze rezultaty dla zdefiniowanego wyżej problemu można uzyskać jedynie poprzez połączenie dwóch najpopularniejszych metod: wykrywania wzorców złośliwego oprogramowania na podstawie łańcuchów tekstowych oraz technikę określania podobieństwa pliku do znanych reprezentantów złośliwego oprogramowania na podstawie ich cech przy pomocy algorytmów klasyfikacji.

Oba podejścia mają swoje wady i zalety: metody sygnaturowe są szybsze w implementacji oraz nie powodują błędów klasyfikacji, natomiast nie pozwalają na wykrywanie zagrożeń które nie były znane twórcy systemu. Jednocześnie wymagają ciągłej i stałej aktualizacji bazy zagrożeń . Metody bazujące na algorytmach z dziedziny sztucznej inteligencji mają cechy uzupełniające w stosunku do pierwszej techniki. Używając tych algorytmów można analizować wirusy które nie były znane twórcom systemu, jednocześnie wymaga to użycia większych zasobów obliczeniowych oraz może prowadzić do zwiększenia liczby błędnych klasyfikacji.

Na rysunku 1 przedstawiono przykładową architekturę modułu implementowanego jako połączenie części sprzętowej oraz oprogramowania.

Źródłem danych wejściowych może być sieć komputerowa lub dysk twardy w zależności od zastosowania. Pierwszym etapem jest wyszukanie w danych wejściowych określonego zbioru n-gramów, obliczenie liczby wystąpień poszczególnych znaków oraz przygotowanie pozostałych atrybutów. Rezultaty tej fazy są

wykorzystywane przez moduł wyszukiwania wzorców oraz moduły klasyfikacji: sprzętowej i programowej. Na wyjściu układu znajduje się moduł zbierający rezultaty wcześniejszych modułów i określający na ich podstawie ostateczną odpowiedź. Moduł klasyfikacji sprzętowej może wykonywać proste w implementacji sprzętowej i wydajne algorytmy klasyfikacji takie jak: drzewa lub reguły decyzyjne [11]. Natomiast jako oprogramowanie można zaimplementować bardziej zaawansowane metody takie jak sieci neuronowe, czy też metody klasyfikacji ze wzmocnieniem. Dzięki temu będzie można połączyć zalety obu modułów: dużą szybkość działania części sprzętowej dla większości przypadków oraz dużą precyzję klasyfikacji części programowej dla trudniejszych przykładów, z którymi część sprzętowa nie będzie sobie radzić.



Rys. 1. Architektura modułu klasyfikacji danych z użyciem akceleracji sprzętowej
Fig. 1. Framework for hardware accelerated data classification

5. Wyniki eksperymentów

5.1. Przygotowanie danych wejściowych

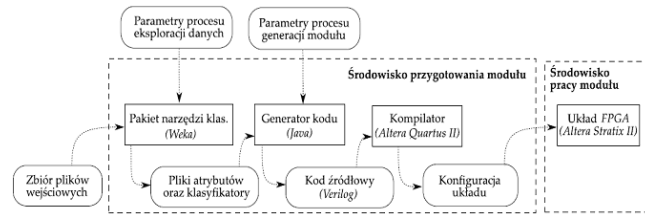
Jako zbiór testowy i trenujący zostały zebrane 6000 plików w następujących formatach:

- pliki wykonywalne w formatach:
  - *Windows Executable,*
  - *Executable Linux Format* (zawierający kod źródłowy jedynie dla architektury *x86*),
  - kod pośredni maszyny wirtualnej języka *Java*,
- skrypty w językach *Perl*, *Bash*,
- dokumenty w formatach:
  - *Portable Document Format*,
  - *Rich Text Format,*
  - *HyperText Markup Language*,
- archiwa kompresowane w formatach:
  - *Zip*,
  - *GNU Zip*,
- dokumenty graficzne w formatach:
  - *Graphics Interchange Format*,
  - *JPEG File Interchange Format.*

Dane wejściowe została zebrane za pomocą programu pająka sieciowego *wget*, używając do tego celu wyszukiwarki *Google*. Można przyjąć założenie, że zebrane w ten sposób dane są w dużej mierze losowe i stanowią dobre odzwierciedlenie danych przesyłanych przez sieć *Internet*.

5.2. Przygotowanie danych wejściowych

Na rysunku 2 przedstawiono sposób przygotowania modułu na podstawie zbioru wstępnie sklasyfikowanych plików wejściowych oraz parametrów całości tego procesu. Wstępna obróbka plików została dokonana za pomocą programu napisanego do tego celu w języku *Java*. Program pobiera pliki podzielone na katalogi zgodnie z ustalonymi klasami i wytwarza opis poszczególnych przykładów w formacie stosowanym przez system *Weka* [4]. Proces przygotowywania wymaga podania parametrów procesu eksploracji danych oraz tworzonego modułu, do których należą: metoda ekstrakcji cech i wybór poszczególnych elementów oraz metoda i parametry klasyfikatora, które są podawane na wejście generatora kodu. Rezultatem pracy generatora jest kod źródłowy w języku opisu sprzętu *Verilog*, który po kompilacji i połączeniu z pozostałymi modułami urządzenia jest kompilowany i symulowany lub ładowany do układu reprogramowalnego *Stratix II* firmy *Altera*.



Rys. 2. Schemat procesu generacji i testowania modułu
Fig. 2. Process of the module generation and testing

5.3. Wyniki dokonanych eksperymentów

Tabele 1, 2 przedstawiają wyniki eksperymentów z użyciem różnych algorytmów ekstrakcji cech oraz klasyfikacji z pakietu *Weka* [4] dla opisanego wcześniej zbioru danych. Przebadane zostały atrybuty: histogramowe (częstości wystąpienia poszczególnych znaków), n-gramowe (informacje o wystąpieniach wybranych wcześniej n-gramów) oraz słownikowe. Tabela 1 przedstawia liczbę komórek logicznych i wynik klasyfikacji dla różnych kombinacji metod ekstrakcji cech oraz klasyfikacji za pomocą drzew decyzyjnych [9]. Spośród n-gramów różnych długości zostały wybrane te o długości dwóch znaków i to one wyróżniają się w tym eksperymencie, uzyskując wysoką precyzję klasyfikacji przy niskim zużyciu komórek logicznych (przeliczanych jako liczba komórek *ALUT* układu *Stratix II* pomnożona przez 1,25 [10]).

Tab. 1. Porównanie różnych modułów ekstrakcji cech
Tab. 1. Comparison of feature extraction modules

Metoda ekstrakcji cech	Liczba komórek logicznych	Precyzja klasyfikacji [%]
Jeden typ atrybutów:		
– znaki	88	72,63
– bi-gramy	67	81,89
– słownik	124	72,84
Dwa typy atrybutów:		
– znaki + bi-gramy	155	82,32
– znaki + słownik	212	76,63
– bigramy + słownik	191	84,84
Trzy typy atrybutów:		
– znaki + bigramy + słownik	279	85,26

Tabela 2 przedstawia wyniki uzyskane przez różne algorytmy klasyfikacji przy atrybutach bigramowych. W tym eksperymencie wyróżniają się metody PART i C45 – pierwsza niskim zużyciem zasobów, a druga najwyższą precyzją klasyfikacji spośród metod sprzętowych. Jak widać, algorytmy meta-klasyfikacji i zaawansowane metody programowe umożliwiają zwiększenie precyzji kosztem użycia procesora lub znacznego zwiększenia ilości bloków mnożących.

Tab. 2. Porównanie różnych modułów klasyfikacji
Tab. 2. Comparison of classification modules

Metoda klasyfikacji	Liczba komórek logicznych	Precyzja klasyfikacji [%]
Sprzętowa klasyfikacja:		
– drzewa decyzyjne (C45)	22	81,89
– reguły decyzyjne (PART)	57	83,79
– reguły decyzyjne (JRIP)	34	81,90
Algorytmy meta-klasyfikacji:		
– losowe drzewa (5 drzew C45)	106 + CPU	86,31
– Ada Boost (3 drzewa C45)	75 + CPU	86,73
– Multi Boost (3 drzewa C45)	53 + CPU	83,97
Klasyfikacja programowa:		
– naiwny klasyfikator Bayesa	CPU	86,95
– Fuzzy Lattice	CPU	87,16
– Hyper Pipes	CPU	87,37

6. Podsumowanie

Ze względu na niskie zużycie zasobów układu reprogramowalnego (poniżej 1% zasobów układów dostępnych obecnie na rynku) oraz wysoką precyzję klasyfikacji danych, uzyskane na tym etapie wyniki można uznać za obiecujące i z całą pewnością stwierdzić, że projekt ten będzie dalej rozwijany, ze względu na wciąż rosnące zapotrzebowanie na tego typu aplikacje.

Urządzenia sieciowe w których ten moduł może być zastosowany to między innymi: systemy przeciwdziałania włamaniom, zapory przeciwogniowe analizujące przesyłane dane, bramy skanujące pocztę elektroniczną, systemy nadzorujące dostęp do stron *WWW*. Moduł zaimplementowany w języku opisu sprzętu może być, podobnie jak biblioteki procedur w przypadku procesorów ogólnego zastosowania, udostępniany jako gotowe rozwiązanie innym projektantom, bez utraty kontroli nad własnością intelektualna, który to sposób dystrybucji określa się powszechnie jako moduły *IPcore* (*Intellectual Property Core*).

7. Literatura

[1] CAI D.M., THEILER J., GOKHALE M.: Detecting a Malicious Executable without Prior Knowledge of Its Patterns. Technical Report LA-UR-03-1205, Los Alamos National Laboratory, 2003.
[2] CICHOSZ P.: Systemy uczące się. Wydawnictwa Naukowo-Techniczne, 2000.
[3] DAMASHEK M.: Gauging Similarity via N-Grams Language-Independent Sorting, Categorization, and Retrieval of Text. Science, 1995, 267, 5199, 843–848.
[4] GARNER S.R.: Weka: The Waikato Environment for Knowledge Analysis. Proceedings of the New Zealand Computer Science Research Students Conference, 57-64, University of Waikato, New Zealand, 1995.
[5] GRIES M.: Algorithm-Architecture Trade-offs in Network Processor Design. Ph.D. Thesis, Swiss Federal Institute of Technology, Zurich, 2001.
[6] KOLTER J., MALOOF M.A.: Learning to Detect Malicious Executables in the Wild. ACM International Conference on Knowledge Discovery and Data Mining, 2004.
[7] LI W., WANG K., STOLFO S., HERZOG B.: Fileprints: Identifying File Types by N-Gram Analysis. Proceedings of the 2005 IEEE Workshop on Information Assurance. United States Military Academy, West Point, NY, USA, 2005.
[8] MCDANIEL M., HEYDARI M.H.: Content Based File Type Detection Algorithms. Proceedings of the 6th Annual Hawaii International Conference on System Sciences, 2003.
[9] QUINLAN R.: C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA, 1993.
[10] MORRIS. K: Morris, "Terminology tango 101: From dog gates to marketing megahertz," FPGA and Programmable Logic Journal, vol. 4, no. 1, 2004.

Artykuł recenzowany