

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Dainius ČEPONIS

RESEARCH OF MACHINE AND DEEP
LEARNING METHODS APPLICATION FOR
HOST-LEVEL INTRUSION DETECTION AND
CLASSIFICATION

DOCTORAL DISSERTATION

TECHNOLOGICAL SCIENCES,
INFORMATICS ENGINEERING (T 007)

Vilnius, 2021

Doctoral dissertation was prepared at Vilnius Gediminas Technical University in 2016–2021.

Supervisor

Assoc. Prof. Dr Nikolaj GORANIN (Vilnius Gediminas Technical University, Informatics Engineering – T 007).

The Dissertation Defense Council of Scientific Field of Informatics Engineering of Vilnius Gediminas Technical University:

Chairman

Prof. Dr Dalius MAŽEIKA (Vilnius Gediminas Technical University, Informatics Engineering – T 007).

Members:

Prof. Dr Habil. Gintautas DZEMYDA (Vilnius University, Informatics Engineering – T 007),

Dr Rytis MASKELIŪNAS (Kaunas University of Technology, Informatics Engineering – T 007),

Dr Jevgenijus TOLDINAS (Kaunas University of Technology, Informatics Engineering – T 007),

Dr Marcin WOZNIAK (Silesian University of Technology, Poland, Informatics – N 009).

The dissertation will be defended at the public meeting of the Dissertation Defense Council of Informatics Engineering in the Senate Hall of Vilnius Gediminas Technical University at **11 a. m. on 13 May 2021**.

Address: Saulėtekio al. 11, LT-10223 Vilnius, Lithuania.

Tel.: +370 5 274 4956; fax +370 5 270 0112; e-mail: doktor@vilniustech.lt

A notification on the intend defending of the dissertation was send on 12 April 2021. A copy of the doctoral dissertation is available for review at Vilnius Gediminas Technical University repository <http://dspace.vgtu.lt>, at the Library of Vilnius Gediminas Technical University (Saulėtekio al. 14, LT-10223 Vilnius, Lithuania) and at the library of Kaunas University of Technology (K. Donelaičio st. 20, LT-44239 Kaunas, Lithuania).

Vilnius Gediminas Technical University book No 2021-008-M

doi:10.20334/2021-008-M

© Vilnius Gediminas Technical University, 2021

© Dainius Čeponis, 2021

dainius.ceponis@vilniustech.lt

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Dainius ČEPONIS

MAŠININIO IR GILAUS APMOKYMO
METODŲ TAIKYMO ĮSILAUŽIMŲ
KOMPIUTERIO LYGMENYJE APTIKIMUI IR
KLASIFIKAVIMUI TYRIMAS

DAKTARO DISERTACIJA

TECHNOLOGIJOS MOKSLAI,
INFORMATIKOS INŽINERIJA (T 007)

Vilnius, 2021

Disertacija rengta 2016–2021 metais Vilniaus Gedimino technikos universitete.

Vadovas

doc. dr. Nikolaj GORANIN (Vilniaus Gedimino technikos universitetas, informatikos inžinerija – T 007).

Vilniaus Gedimino technikos universiteto Informatikos inžinerijos mokslo krypties disertacijos gynimo taryba:

Pirmininkas

prof. dr. Dalius MAŽEIKAS (Vilniaus Gedimino technikos universitetas, informatikos inžinerija – T 007).

Nariai:

prof. habil. dr. Gintautas DZEMYDA (Vilniaus universitetas, informatikos inžinerija – T 007),

dr. Rytis MASKELIŪNAS (Kauno technologijos universitetas, informatikos inžinerija – T 007),

dr. Jevgenijus TOLDINAS (Kauno technologijos universitetas, informatikos inžinerija – T 007),

dr. Marcin WOZNIAK (Silezijos technologijos universitetas, Lenkija, informatika – N 009).

Disertacija bus ginama viešame Informatikos inžinerijos mokslo krypties disertacijos gynimo tarybos posėdyje **2021 m. gegužės 13 d. 11 val.** Vilniaus Gedimino technikos universiteto senato posėdžių salėje.

Adresas: Saulėtekio al. 11, LT-10223 Vilnius, Lietuva.

Tel.: (8 5) 274 4956; faksas (8 5) 270 0112; el. paštas doktor@vilniustech.lt

Pranešimai apie numatomą ginti disertaciją išsiųsti 2021 m. balandžio 12 d.

Disertaciją galima peržiūrėti Vilniaus Gedimino technikos universiteto talpykloje <http://dspace.vgtu.lt>, Vilniaus Gedimino technikos universiteto bibliotekoje (Saulėtekio al. 14, LT-10223 Vilnius, Lietuva) ir Kauno technologijos universiteto bibliotekoje (K. Donelaičio g. 20, LT44239 Kaunas, Lietuva).

Abstract

The growing number of intrusions on the information systems (in the context of this thesis, information system is understood as a host i.e. end system such as a personal computer, server, etc. but not the network equipment or traffic) level requires more sophisticated methods to combat cyber-attacks. A significant loss for companies is expected if those actions are not recognised and averted. The intrusion detection systems (IDS) and antivirus applications (AV) are the main approaches to combat host-level cyber-attacks. The problem is that most popular solutions are utilising signature-based methods which are incapable of detecting new and emerging attacks.

Anomaly-based IDS are designed to detect zero-day attacks with much higher accuracy than signature-based. Typically, these systems utilise statistical or machine learning methods to perform the task. Unfortunately, an enormous amount of data is required to train and validate anomaly-based systems. Currently, host-based intrusion detection systems (HIDS) lack such data in comparison with network-based systems (NIDS) and a new extensive dataset would contribute to host-based intrusion detection research.

This dissertation consists of an introduction, four main chapters and general conclusions. The first chapter introduces existing intrusion and malware detection methods as well as approaches to data collection techniques. Existing datasets, machine learning (ML) and deep learning (DL) methods, currently used in HIDS, are reviewed at the end of chapter one. The second chapter proposes a robust method of dataset generation of malicious activity for anomaly-based HIDS training as well as introduces the generated Attack-Caused Windows System Calls Traces Dataset (AWSCTD) and its characteristics. The third chapter investigates ML methods applicability in intrusion and malware detection with the newly presented host-level dataset. Chapter four discusses the application of vanilla and advanced DL methods trained with the newly generated dataset: the new DL models are proposed and compared with already recognised state-of-the-art models.

The experiments and analysis performed have demonstrated that the utilisation of virtualisation technologies allows the effective automation of dataset generation in cases where data-generating systems should be securely isolated. Simple ML methods are not sufficient for the host-level and malware detection task compared to DL methods due to comparatively low (90–92%) accuracy. Furthermore, the proposed static single-flow DL model outperformed already recognised state-of-the-art models in the intrusion detection task. Lastly, the sequence of 600 first system calls from Windows applications allows achieving more than 95% detection accuracy that is enough to perform the majority of anomaly-based intrusion and malware detection tasks adequately.

Reziumė

Augantis kompiuterio lygmens įsibrovimų kiekis reikalauja vis sudėtingesnių metodų panaudojimo prieš kibernetines atakas. Jei metodai nėra atnaujinami – įmonės patiria didelius nuostolius. Kompiuterio lygmens kibernetinių atakų aptikimui naudojamos įsibrovimo aptikimo sistemos ir antivirusinės programos. Parašais paremtų metodų naudojimas populiariausiuose produktuose lemia tai, kad neaptingamos naujos ir populiarėjančios atakos.

Anomalijomis paremtos įsibrovimo aptikimo sistemos yra suprojektuotos aptikti nulinės dienos atakas didesniu tikslumu, nei parašais paremti metodai. Standartiškai, savo darbui atlikti, šios sistemos naudoja statistinius arba ML (angl. *Machine Learning*) paremtus metodus, tačiau šių sistemų apmokymui ir validacijai reikalingi dideli duomenų kiekiai. Naujas išsamus duomenų rinkinys padėtų kompiuterio lygmens įsibrovimų aptikimo tyrimams, nes šiuo metu kompiuterio lygmens įsibrovimo aptikimo sistemos turi nepakankamą kiekį duomenų palyginus su tinklo lygmens sistemomis.

Disertaciją sudaro įvadas, keturi pagrindiniai skyriai ir bendrosios išvados. Pirmame skyriuje pristatomi įsibrovimų ir kenksmingo programinio kodo aptikimo metodai informacinėse sistemose bei aptariami egzistuojantys kompiuterio lygmens duomenų rinkiniai, skirti anomalijomis paremtų sistemų apmokymui. Pirmo skyriaus pabaigoje aprašomi ML ir DL (angl. *Deep Learning*) metodai naudojami kompiuterio lygmens įsibrovimų aptikimui. Antrame skyriuje pristatomas patikimas būdas anomalijomis paremtų įsibrovimo sistemų apmokymui skirtu duomenų rinkinio kūrimas, bei aprašomas surinktas AWSCTD (angl. *Attack-Caused Windows System Calls Traces Dataset*) duomenų rinkinys ir jo parametrai. Trečiame skyriuje įvertinamas klasikinių ML metodų pritaikomumas įsibrovimo ir kenksmingo programinio kodo aptikimui panaudojant AWSCTD. Ketvirtame skyriuje aptariamas paprastų ir sudėtingų DL metodų panaudojimas apmokant juos su AWSCTD.

Atlikti eksperimentai ir jų analizė parodė, kad virtualizacijos technologijų panaudojimas leidžia efektyviai automatizuoti duomenų rinkinio kūrimą, kai naudojamos sistemos turi būti saugiai izoliuotos. Dėl žemo tikslumo (90–92 %) paprasti ML metodai yra neperspektyvūs atlikti kompiuterio lygmens įsibrovimo ir kenksmingo programinio kodo aptikimo užduotims, palyginus su DL metodais. Be to, įsibrovimo užduotyje pasiūlytas statinis vieno srauto DL modelis pademonstravo geresnius rezultatus, nei jau pripažinti modeliai. Galiausiai, pirmųjų 600 sisteminių funkcijų sekos iškviestos Windows operacinėje sistemoje, leidžia pasiekti didesnę nei 95% aptikimo tikslumą, kuris yra daugiau nei pakankamas tinkamam didžiosios dalies anomalijomis paremto įsibrovimo ir kenksmingo programinio kodo aptikimo užduoties atlikimui.

Notations

Abbreviations

AI – artificial intelligence;

AIDS – anomaly-based intrusion detection system;

ANN – artificial neural network;

AV – anti-virus;

AWSCTD – attack-caused windows OS system calls traces dataset;

BPTT – backpropagation through time;

BSM – basic security module;

CNN – convolutional neural networks;

DBN – deep belief network;

DL – deep learning;

EER – effective error rate;

FAR – false alarm rate;

FNR – false negative rate;

FPR – false positive rate;

HIDS – host-based intrusion detection system;

HMM – hidden Markov model;

ICS – industrial control systems;

ID – intrusion detection;

IDES – intrusion-detection expert system;

IDS – intrusion detection system;
LSTM – long short-term memory;
MAE – Mean absolute error;
MCC – Mathews correlation coefficient;
ML – machine learning;
MLP – multi layer perceptron;
NIDS – network-based intrusion detection system;
NN – neural network;
NSM – network security monitor;
RMSE – root mean square error;
RNN – recurrent neural networks;
ROC – receiver operating characteristic;
SCADA – supervisory control and data acquisition systems;
SIDS – signature-based intrusion detection system;
SONN – self organizing neural networks;
SVM – support vector machines.

Contents

INTRODUCTION	1
Problem Formulation.....	1
Relevance of the Thesis.....	2
The Object of Research	2
The Aim of the Thesis	2
The Tasks of the Thesis	3
Research Methodology	3
Scientific Novelty of the Thesis	4
Practical Value of the Research Findings.....	4
Defended Statements.....	5
Approval of the Research Findings	5
Structure of the Dissertation	6
1. LITERATURE REVIEW ON HOST-LEVEL INTRUSION AND MALWARE DETECTION TECHNIQUES ON THE INFORMATION SYSTEMS.....	7
1.1. Current Tendencies in Malware and Host-Level Intrusion Prevalence	8
1.2. The Introduction to Intrusion and Malware Detection Systems on Host-Level	9
1.2.1. Intrusion Detection System Location.....	11
1.2.2. Analysis Strategy of Detection	17
1.2.3. Events Prioritisation Problem in Intrusion and Malware Detection Systems.....	20
1.2.4. Techniques Used in Host-Based Intrusion and Malware Detection.....	20
1.3. Anomaly-based Intrusion and Malware Detection	22

1.3.1. Anomaly-Based Techniques of Malware and Intrusion Detection	26
1.3.2. Application of Machine Learning Methods in Anomaly-based Intrusion and Malware Detection.....	28
1.3.3. Datasets Designed for Malware and Intrusion Detection.....	37
1.3.4. Application of Artificial Neural Networks and Deep Learning in Intrusion and Malware Detection Systems	43
1.4. Application of System Calls for Intrusion and Malware Detection.....	52
1.5. Intrusion and Malware Detection Research Situation in Lithuania	55
1.6. Conclusions of Chapter 1 and Formulation of the Objectives of the Thesis	57
 2. GENERATION OF HOST-BASED WINDOWS DATASET FOR ANOMALY- BASED INTRUSION DETECTION	59
2.1. Dataset Generation Method Description	59
2.2. Dataset Generation Method Implementation.....	61
2.3. The Process of Malware Loading to the Guest Machine	63
2.4. Attack-Caused Windows System Calls Traces Dataset.....	65
2.4.1. Dataset Size and Structure	66
2.4.2. Dataset Characteristics.....	71
2.5. AWSCTD Dataset Balancing	73
2.6. Methods for Minimising Overfitting Problem.....	74
2.7. Classification Methods Evaluation Criteria.....	75
2.8. Conclusions of Chapter 2	78
 3. EVALUATION OF MACHINE LEARNING METHODS APPLICATION FOR ANOMALY-BASED INTRUSION DETECTION AND CLASSIFICATION.....	81
3.1. Application of AWSCTD for the Malware Classification.....	82
3.2. Experimental Testbed Configuration and Selected Criteria	82
3.3. Feature Pre-Processing and Selected ML Methods	83
3.4. Results of Malware Classification	85
3.5. Conclusions of Chapter 3	90
 4. EVALUATION OF DEEP LEARNING METHODS APPLICATION FOR ANOMALY-BASED INTRUSION DETECTION AND CLASSIFICATION.....	93
4.1. Motivation of Dataset and DL Models Selection	94
4.2. Feature Processing.....	95
4.3. Experimental Testbed Configuration.....	98
4.4. Evaluation of Vanilla DL Models	98
4.4.1. Model Selection Criteria and Configuration Parameters	98
4.4.2. Evaluation of Malware Classification Task	100
4.4.3. Evaluation of the Family Classification Task	103
4.4.4. Evaluation of the Intrusion Detection Task	106
4.5. Evaluation of Advanced DL Models	112
4.5.1. Advanced DL Models Used and Configuration Parameters	112
4.5.2. Evaluation of the Intrusion Detection Task	116
4.6. Conclusions of Chapter 4	130

GENERAL CONCLUSIONS	133
REFERENCES	137
LIST OF SCIENTIFIC PUBLICATIONS BY THE AUTHOR ON THE TOPIC OF THE DISSERTATION.....	155
SUMMARY IN LITHUANIAN.....	157
ANNEXES ¹	175
Annex A. The Result Tables of Deep Learning Experiments	175
Annex B. Declaration of Academic Integrity	179
Annex C. The Co-Authors' Agreements to Present Publications Material in the Dissertation.....	180

¹ The annexes are supplied in the enclosed compact disc.

Introduction

Problem Formulation

The increasing number of cyber-attacks requires an adoption of more sophisticated methods for host-level malware and intrusion detection. The widely used method of signature-based intrusion detection is failing to detect zero-day and emerging attacks. In addition, attacks specially designed for the particular target are also undetectable (Xie & Hu, 2013).

Another problem is that network-based intrusion detection systems are dominating the field and contain a sufficient number of datasets for training anomaly-based intrusion detection methods. Host-based intrusion and malware detection systems, on the other hand, are lacking datasets of good quality. However, HIDS and AV systems are receiving more attention due to the fact that they provide more information about intrusion and can prevent from significant damage (e.g., alteration of critical system files) as well as offering an additional layer of security (Kozushko, 2003). Furthermore, the existing HIDS datasets are mostly covering a Linux OS and are missing Windows OS – the most popular operating system for desktop machines according to the statistics (StatCounter Global Stats, 2019).

Anomaly-based methods are helping to improve cyber-attack detection rates, especially for zero-day and polymorphic malware attacks. The main idea of anomaly-based methods is to retrieve a standard behaviour profile from a user or system

and mark non-expected actions as an intrusion (with the respect of threshold values) (Aldweesh et al., 2020).

In recent years, after computer hardware has been improved (especially graphics cards) – deep learning (DL) methods have been used to increase anomaly-based intrusion detection systems accuracy and lower false alarm rate (Hwang et al., 2019).

Relevance of the Thesis

Researchers studying intrusion and malware detection systems are using complicated machine learning models to achieve characteristics that can be acceptable in real-world usage. This approach has two problems: these models require an enormous amount of data for training, and they use a significant amount of computer resources.

Additionally, the abundance of network-based datasets is steering researchers to investigate only one aspect of the problem. Malicious actions detection on one layer of an information system can lead to an undetected cyber-attack in another one, i.e. host level. Most existing datasets suggest that intrusion can only come from network and Linux OS-based computers. On the contrary, Windows OS-based systems are still the most popular ones on end-user machines.

The dataset problem can be solved straightforwardly – a novel robust method for such a dataset generation must be proposed and implemented. Creation of a dataset provides a basis for further research on ML and DL methods effectiveness for host-level intrusion and malware detection task.

Research performed utilising the proposed AWSCTD dataset on the effectiveness of different ML and DL methods and models has allowed identifying the most effective approaches for further use in practical system implementation and directions for further research.

The Object of Research

The object of the present study is the application of ML and DL methods for host-level intrusion and malware detection in information systems.

The Aim of the Thesis

The aim of the research is to improve anomaly-based host-level intrusion and malware detection metric values by applying ML and DL methods.

The Tasks of the Thesis

In order to achieve the aim of the thesis, the following tasks have to be solved:

1. To perform a scientific literature review on the current state of ML/DL methods and available Windows OS datasets used for host-level intrusion and malware detection.
2. To propose a robust method for generating a comprehensive dataset of classified actions by malicious and benign applications that can be used as a basis for further research on ML and DL methods effectiveness for host-level intrusion and malware detection task.
3. To perform experimental evaluation and applicability analysis on the basis of the identified metrics for the host-level intrusion and malware detection task of the following data analysis methods:
 - 3.1. Existing most widely used ML methods;
 - 3.2. Existing most widely used vanilla DL methods;
 - 3.3. Existing and newly proposed/modified advanced DL methods.

Research Methodology

The quantitative methodology has been used for this thesis. In order to get better insight into the possibilities for improvement of the intrusion and malware detection, literature review of current methods and used datasets have been conducted with focus on host-based field. Later, several experiments have been executed to collect novel dataset for intrusion and malware detection and determine best performing ML and DL methods. The HP ProLiant DL 380 G6 server and author proposed methods have been used for novel dataset generation. The computer with Intel i7-7500U 2.70 GHz processor, 8 GB RAM and Windows 10 operating system and Weka 3.8 software have been used to determine best ML methods for malware classification. The computer with Intel i5-3570 3.80 GHz processor, GTX 1070 GPU, 16GB RAM, Ubuntu 18.04.3 LTS operating system and libraries of Keras and Tensorflow have been used to determine best DL methods for intrusion and malware detection. Data gathered was compared with typical ML and DL metrics: accuracy, precision, recall, f-1 score, false positive rate, false negative rate and classification error.

Scientific Novelty of the Thesis

The scientific novelty of this study is specified as follows:

1. A novel host-based dataset (AWSCTD) was created in order to be used for anomaly-based intrusion and malware detection in Windows operating system. It includes a vast number of system calls of malicious and benign applications and has an advantage against currently existing datasets by amount and type of data collected. The dataset was created using the proposed robust method developed by the author of the thesis.
2. A novel approach to use existing vanilla dynamic CNN-based model for the classification of malicious and benign applications by sequence calls as the feature vector has been proposed. The existing dynamic CNN-based model, identified experimentally, outperforms more advanced CNN-based and sequence-based RNN models used by other researchers in almost all evaluation metrics or allows achieving similar results while demonstrating better time consumption rates. The accuracy achieved for malicious and benign applications classification has reached 99.1–99.3% when using AWSCTD dataset for model training and using 1000 of subsequent system calls.
3. A novel previously unused single-flow static CNN-based model has been proposed for anomaly-based intrusion and malware detection by using system calls generated by applications as the feature vector. The proposed method outperforms the state-of-the-art dual flow static models in terms of accuracy, training and classification time when using AWSCTD dataset.

Practical Value of the Research Findings

The achieved results are significant both from the theoretical and practical points of view for the crucial task of anomaly-based intrusion detection in information systems as today more and more cyber-attacks are emerging, and are specifically targeted and conventional signature-based methods are no longer working.

The novel extensive AWSCTD dataset produced will allow researchers to continue experiments for the improvement of anomaly-based HIDS as currently a significant number of datasets are created for NIDS. The current datasets targeted at HIDS are limited and lack essential features.

The achieved accuracy of 99.1–99.3% and classification time needed (0.00223s) to perform the malicious/benign classification task, i.e. attack or malware detection, insures practical reliability and performance rates for the proposed methods and models applicability in commercial IDS and AV systems in possible combination with classical signature-based methods, where the latter would be used for reliable known attack classification, while anomaly-based methods (vanilla dynamic CNN-based model and novel single-flow static CNN-based model) would be used for non-identified suspicious cases.

Defended Statements

The following statements, based on the results of the present thesis, may serve as the official hypotheses to be defended:

1. ML methods are less effective for the host-level and malware detection task compared to DL methods due to comparatively low (90–92%) accuracy when subsequent system calls generated by malware are used for classification task.
2. Vanilla models outperform complex state-of-the-art dual-flow static deep learning models on univariate time series data used for host-level intrusion and malware detection task, when AWSCTD is used for model training.
3. The sequence of 600 foremost system calls from Windows applications allows achieving more than 95% detection accuracy that is enough to allow the application of identified or proposed DL techniques in hybrid or enterprise-oriented security solutions that combine automatic detection of the major part of anomalies leaving unclear cases for human-expert, meanwhile full automation of the detection process and 99.3% accuracy can be ensured by using the first 1000 system functions.

Approval of the Research Findings

The results of the dissertation were published in 6 scientific publications. 3 of them are published in *Clarivate Analytics* (also referred to as *Thomson Reuters*) *Web of Science* databases refereed journals. 3 are published in conference proceedings. The author has also made 6 presentations at international and domestic scientific conferences:

- The 13th Junior Researchers Conference “Science – Future of Lithuania” on IT Security and Information Systems, 2010, Vilnius, Lithuania.

- The 20th Junior Researchers Conference “Science – Future of Lithuania” on IT Security and Information Systems, 2017, Vilnius, Lithuania.
- The 13th International Baltic Conference On Databases And Information Systems (Baltic DB&IS 2018), Trakai, Lithuania.
- The second International Scientific Conference on Security (CONFSEC 2018), Borovets, Bulgaria.
- The 24th International Nordic Conference on Secure IT Systems (NordSec 2019), Aalborg, Denmark.
- The 11th International Workshop Data Analysis Methods for Software Systems (DAMSS 2019), Druskininkai, Lithuania.

Structure of the Dissertation

The scientific work consists of an introduction of the dissertation, four main chapters, general conclusions, references, list of author’s publications and annexes. The total scope of the dissertation – 156 pages without annexes. There are 54 pictures and 47 tables in the text. 228 references were used in the dissertation text.

Literature Review on Host-Level Intrusion and Malware Detection Techniques on the Information Systems

This chapter reviews techniques used to detect cyber-attacks against information systems and the importance of intrusion detection. The primary definition of intrusion detection is given. An explanation is provided on the intrusion detection task implementation and operation cases, determined by the IDS placement on the network level, end-user machines or application of a hybrid approach. Various methods of data analysis for intrusion detection such as statistical, machine learning and deep learning are discussed with a comprehensive analysis of datasets used to develop such methods. This chapter concludes with the clarification of the main objective and tasks of the present investigation, as well as a summary of the literature review findings.

Analysis presented in this Chapter was published in (Čeponis & Goranin, 2018; Goranin & Čeponis, 2018; Čeponis & Goranin, 2020b).

1.1. Current Tendencies in Malware and Host-Level Intrusion Prevalence

A cyber-attack is often described as any kind of malicious activity that attempts to collect, disrupt, deny, degrade, or destroy information system resources or the information itself. As stated in the report of the year 2019 by the National Cyber Security Centre of Lithuania, cybercrimes increased three times compared to the previous period (Krašto apsaugos ministerija, 2020). Intrusion to information systems and propagation of malicious applications was the most popular type of incidents. On the global scale, according to a 2019 Symantec Internet Security Threat Report – 23% of attack groups were using zero-day vulnerabilities. Additionally, ransomware infections in 2018 have grown by 12% for the enterprises (Symantec, 2019). The Data Breach report of 2019 is stating that globally, 60% of companies say they have been breached at some point in their history, with 30% having experienced a breach within the past year alone. In the U.S., the numbers are even higher, with 65% ever experiencing a breach, and 36% within the past year (Thales, 2019). One of the latest incidents: LifeLabs Medical Laboratory Services, Canada's largest lab testing company, has paid a ransom after a major cyberattack led to the theft of lab results of 85,000 Ontarians and potentially the personal information of 15 million customers (Kiladze, 2019). Industrial control systems (ICSs) are another critical attack vector targeted by the cyber-criminals (Gonda, 2014). In recent years there have been several important incidents in global ICSs. With the help of operating system vulnerabilities, an Iranian nuclear power station was attacked by Stuxnet in 2010 (Yang et al., 2014). Stuxnet, as it came to be known, was unlike any other virus or worm that came before. Rather than simply hijacking targeted computers or stealing information from them, it escaped the digital realm to wreak physical destruction on equipment the computers controlled (Radziwill, 2018). Later, in 2014, the ICSs in the energy field of Europe and the United States have been attacked by the Havex. It has been used to target supervisory control and data acquisition (SCADA) systems. In this incident, three ICS vendor websites were compromised, so miscreants could replace legitimate software installers hosted on sites with ones that had a trojan Havex RAT integrated (Walker, 2014). There is no doubt that to run critical infrastructure or enterprise steadily and reliably intrusion and malware detection systems are one of the most valuable critical tools. It can prevent from data loss, downtime and damaged equipment as stated above.

1.2. The Introduction to Intrusion and Malware Detection Systems on Host-Level

Intrusion detection systems (IDS) is a system dedicated for monitoring user and application activity on a company network and computers and alerting of possible attack attempts. The more precise definition of the intrusion detection system is provided by (Bace & Mell, 2001) and defines IDS as “the process of monitoring the events occurring in a Computer system or network and analysing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network”.

Malware is a collective term for any malicious software which enters the system without the authorisation of the user of the system. The term malware comes from combining the two words malicious and software, and to be used to indicate any unwanted software. It was defined, by G. McGraw and G. Morrisett as “any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system” (McGraw & Morrisett, 2000). A malware detection application/system (detector) is usually placed on the end-user machine (host-level). A malware detection program is a computational function that works in a domain which contains a collection of application programs, and a collection of malicious and benign programs. The detector program analyses the programs which belong to the set of application programs to find whether it is a benign (normal program), or a malware (malicious program).

An architectural framework of a generic intrusion detection system is shown in Figure 1.1. The system generally contains, but is not limited to, the following components (Lazarevic et al., 2005):

- *Data gathering device (sensor)* is responsible for collecting data from the monitored system.
- *Detector (Intrusion Detection (ID) analysis engine)* processes the data collected from sensors to identify intrusive activities.
- *Knowledge base (database)* contains information collected by the sensors, but in pre-processed format (e.g. knowledge base of attacks and their signatures, filtered data, data profiles, etc.). This information is usually provided by network and security experts.
- *Configuration device* provides information about the current state of the intrusion detection system (IDS).
- *Response component* initiates actions when an intrusion is detected. These responses can either be automated (active) or involve human interaction (inactive).

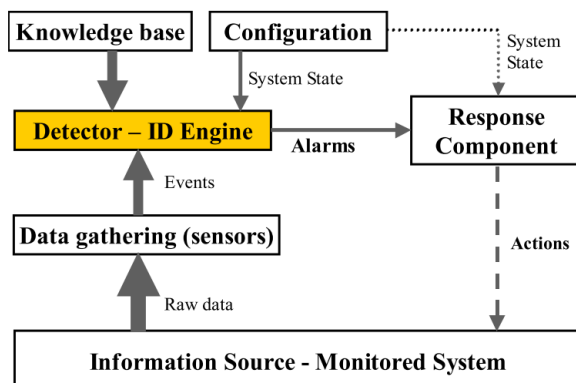


Fig. 1.1. Generic architecture of intrusion detection system (Lazarevic et al., 2005)

In 2005 already mentioned Lazarevich and his colleagues undertook research on intrusion detection systems taxonomy. They combined previous attempts (produced in 1999 and 2000) and proposed a taxonomy to classify intrusion detection systems according to their information source, analysis strategy, time aspects, architecture and response types (Axelsson, 2000; Lazarevic et al., 2005). Proposed taxonomy of IDSs can be seen in Figure 1.2.

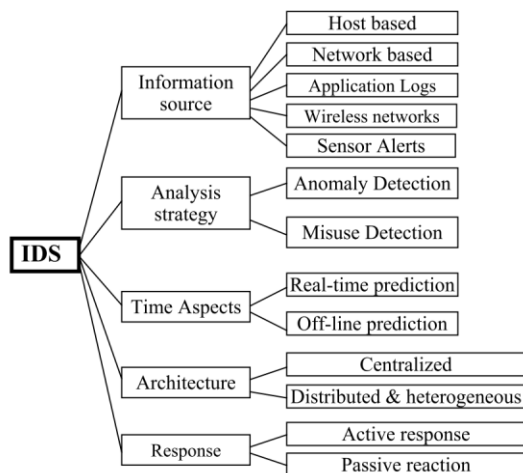


Fig. 1.2. Taxonomy of intrusion detection systems according to proposed six criteria (Lazarevic et al., 2005)

The first criteria are the *source of the data* because the data is the primary source of information on actions executed on the monitored system. The primary sources of data are as follows:

- Audit trails (system logs) on host;
- Network packets/connections;
- Applications logs;
- Wireless network traffic;
- Other sensors or intrusion detection systems alerts;

The *analysis strategy* defines how does decision is evaluated by IDS. If IDS is looking for already defined actions, events or log records – it is using a method of misuse or signature detection. If IDS is looking for abnormal events, traffic or behaviour in the monitored system – it is using an anomaly-based strategy.

The *time aspects* are used to categorise the IDSs into on-line IDSs that detect intrusions in real-time and off-line IDSs that usually first store the monitored data and then analyse it in batch mode for signs of intrusion.

The *aspect of architecture* points to audit data collection method: data can be collected from many different sources in a distributed fashion, or from a single point using the centralised approach. The former method has one advantage – it can provide more information on global and coordinated attacks.

The *IDS response* can be either reactive or proactive. Reactive IDS only log the exact attack event with all required information (time, location, affected system parts and etc.) and notifies security personnel. The proactive system is designed to execute specific tasks on the attack: logging out possible attackers, closing down services or closing security holes. Proactive systems are called intrusion prevention systems.

1.2.1. Intrusion Detection System Location

The capabilities of IDS largely depend on the data it has available (Modi et al., 2013), and for that reason, location is a critical architectural decision. Despite the aforementioned IDS classification by data source criteria, the network and host located IDS classification can be coarsely used as two main types: Host-based IDS (HIDS) and Network-based IDS (NIDS). HIDS inspect data that originates from the host system and audit sources, such as operating system, window server logs, firewalls logs, application system audits, or database logs (Creech & Hu, 2014). However, there are IDSs that utilise both network and host data, usually with the goal of achieving more complete visibility of a host. They are called hybrid (or distributed) intrusion detection systems and usually provides better attack scope (Y. Lin et al., 2010). For example, in one of the earliest intrusion detection systems developed by (Snapp et al., 1991), Haystack (Smaha, 1988) was used on each host

to detect local attacks, while network security monitor (NSM) (Heberlein et al., 1990) was utilised to monitor the network. Both systems, Haystack and NSM, send information to the DIDS Director, where the final analysis is performed.

With the rapidly growing popularity of the Internet services, there has been an increasing number of attacks aimed at the network itself (e.g. spoofing, TCP hijacking, port scanning, ping of death) that cannot be (at least not easily) detected by examining the host audit trail alone. These reasons have led to the development of specific tools that sniff network packets in real-time and facilitate searching for network attacks (Lazarevic et al., 2005). NIDS monitors the network traffic that is extracted from a network through packet capture, NetFlow, and other network data sources. Network-based IDS can be used to monitor many computers that are joined to a network. NIDS is able to monitor the external malicious activities that could be initiated from an external threat at an earlier phase, before the threats spread to another computer system. On the other hand, NIDSs have limited ability to inspect all data in a high bandwidth network because of the volume of data passing through modern high-speed communication networks (Bhuyan et al., 2015). NIDS are deployed at a number of positions within a particular network topology, together with HIDS and firewalls, can provide a concrete, resilient, and multi-tier protection against both external and insider attacks.

The basic components of a generic anomaly-based NIDS system architecture are listed below (Bhattacharyya & Kalita, 2013):

- *Traffic Capture*: Traffic capturing is an important module in any NIDS. The raw traffic data is captured at both packet and flow levels. Packet level traffic can be captured using a common tool, e.g., Gulp, Wireshark, tcpdump and then pre-processed before sending to the detection engine. Flow level data, in case of the high-speed networks, is composed of information summarised from one or more packets. Some common tools to capture the flow level network traffic include Nfdump, NfSen and Cisco Netflow.
- *Anomaly Detection Engine*: This is the heart of any network anomaly detection system. It attempts to detect the occurrence of any intrusion, either online or offline. In general, any network traffic data needs pre-processing before it is sent to the detection engine. If the attacks are known, they can be detected using a misuse detection approach. Unknown attacks can be detected with the anomaly-based approach using an appropriate matching mechanism.
- *Reference Data*: The reference data stores information about signatures or profiles of known intrusions or normal behaviour.
- *Configuration Data*: Intermediate results such as partially created intrusion signatures are stored as configuration data.

- *Alarm*: This component of the architecture is responsible for the generation of an alarm based on the indication received from the Detection Engine.
- *Post-Processing*: This is an important module that processes the generated alarms for diagnosis of actual attacks. Appropriate post-processing activities can help reduce the false positive rate significantly.
- *Security Manager*: Stored intrusion signatures are updated by the Security Manager (SM) as and when new intrusions become known.

It is worth mentioning one of the main NIDS disadvantages. If the incoming traffic is encrypted, then the detection engine is rendered useless because it cannot search for patterns of misuse in the payload of network packets. For example, attacks on encrypted protocols such as HyperText Transfer Protocol Secure (HTTPS) cannot be read by an IDS. The IDS cannot match the encrypted traffic to the existing signatures if it doesn't interpret the encrypted traffic. Therefore, examining encrypted traffic makes it difficult for detectors to detect attacks (Butun et al., 2014). For example, packet content-based features have been applied extensively to identify malware from regular traffic, which cannot be readily applied if the packet is encrypted. This challenge encourages investigators to use statistical network flow features, which do not rely on packet content (Camacho et al., 2016). As a result of this, attacks through an encrypted network can potentially be identified from normal traffic.

The elite IDSs detect as much malicious traffic as possible and reduce the number of false alarms. There are a number of commercial IDSs available in the market such as Juniper, McAfee, Cisco, Symantec etc. (Scarfone et al., 2007). The commercial IDS generally do not provide the ideal performance as advertised and could compromise computer network security (Shah & Issac, 2018). Like the commercial IDSs, there are a number of open-source IDSs available such as Snort, Suricata and Zeek (formerly Bro) (Tirumala et al., 2015).

Host-based intrusion system (HIDS) is located on endpoint user machine and monitors user and host operating system behaviour and was the first type of IDS. A generic HIDS architecture is similar to NIDS provided in Figure 1.3. only internet traffic capture module is replaced with a host system data capture module (as stated before – this taxonomy is based on data source). A HIDS system will require some software that resides on the system and can scan all host resources for activity; some just scan syslog and event logs for activity. It will log any activities it discovers to a secure database and checks to see whether the events match any malicious event record listed in the knowledge base (Grimmer et al., 2018). HIDS can use the following data sources to perform its task:

- *System logs and system audit data*: These records keep information on operating system generated records that contain information about occurred events: warnings, errors, system failures, etc. (Rice & Borgman, 1983). Logs can also be produced by the applications executed on the operating system. Application logs can contain information about user sessions, such as login time, user-program interactions, authentication result, etc. Logs produced by the operating system are typically called system logs, while log records produced by the applications are usually referred to as audit data (W. Lee et al., 1998). The performance of the target system is the main challenge when analysing log/audit data: this task is CPU intensive and typically requires human expertise. Furthermore, log analysis requires that all performed actions by the OS are stored and only then feature extraction and classification can be executed. Several solutions have been proposed on how to improve typical log-based systems. For example, Reuning suggested an anomaly-based approach with Bayesian probability theory and the term frequency inverse document frequency (TF-IDF) information retrieval technique (Reuning, 2004). The results achieved with DARPA99 dataset show a high false-positive rate and many undetected attacks. The conclusion of such research was that the proposed method could be a valuable component of a larger IDS. Tchakoucht et al. conducted an experiment with a health information system consisting of three users, a patient, doctor, and an administrator, including their behaviour over 30 days (Ait Tchakoucht et al., 2015). The results generated a sizable increase in successfully identifying users and a large decline in false positives in comparison with previous similar work. Two constraints have been identified: when the user behaviour changes significantly over time, it produces an increase of false-positive rate and the method may not work properly when the data is huge (huge number of users and for a long duration of audit time). One of the latest research papers by Wang & Zhu propose a centralised HIDS model for the private cloud computing environment based on OpenStack (Zhijian Wang & Zhu, 2017). Comparing the new centralised HIDS with a traditional HIDS shows that it achieves better performance in reducing CPU and Memory utilisation, in the case of ensuring detection accuracy and real-time. Mehnaz & Bertino presented a HIDS that profiles users based on their file-system access patterns and detects anomalies (Mehnaz & Bertino, 2020). The mentioned solution has 98.7% accuracy in identifying malicious users files access;
- *System calls*: System call tracing is a popular source of data for intrusion detection as unusual system call sequences can produce evidence of po-

tential irruptions. System calls as the data source for intrusion and malware detection nowadays are more popular than log files. The main reason for that is that they depict primary data produced by the applications (Creech & Hu, 2014) (both legal and malicious), and there is no filtering, interpretation, nor processing applied. Application of system calls for the intrusion detection task is widely discussed, and its popularity is increasing as new training datasets and machine learning (ML) or DL methods arise (Waqas Haider et al., 2016; Sharafaldin et al., 2018; Tran et al., 2018). As system calls are one of the richest information sources for HIDS, it had been selected as the main thesis point of interest. For that reason the review of system calls usage research is moved to a separate section;

- *Windows registry*: Windows operating system configuration settings for all programs and hardware on that host are placed in Windows registry. All processes use the registry to achieve their goals. Since malware also falls into that scope (Monika et al., 2016), Windows registry analysis can be seen as a source for analysis by anomaly-based HIDS, while the majority of existing HIDS solutions simply monitor and react to modifications made to the registry (Hay et al., 2008). Contrarily, in 2002, Apap et al. proposed a Registry Anomaly Detection (RAD) system (Apap et al., 2002). They have shown that registry activity is regular, and described ways in which attacks would generate anomalies in the registry. This type of intrusion detection would improve protection from zero-day attacks. Later, two papers proposed to use one-class support vector machines for registry anomaly detection (Heller et al., 2003; Stolfo et al., 2005). Unfortunately, they both concluded that RAD with the probabilistic approach of Apap et al. is much more accurate. Topallar et al. proposed to use Self-Organising Maps for RAD systems (Topallar et al., n.d.). The results of this study show that the proposed system is effective in detecting the behaviour of malicious software and has a low rate of false alarms compared to other host-based intrusion detection systems;
- *File system*: File system changes are typically made by intruders (changes to configuration files, the introduction of new users, cleaning of log files to hide the intrusion traces, etc.) and malicious applications (code hiding, installation files, etc.). One of the most popular HIDS that monitors changes in critical files is Open Source Host-based Intrusion Detection System (OSSEC) (Hay et al., 2008; Lhotsky, 2013). It runs both on Windows and Linux operating systems as agent and reports file changes to the central server. Another notable HIDS is Samhain and Open Source Tripwire. Samhain provides file integrity checking and log file monitoring/analysis, as well as rootkit detection, port monitoring, detection of

rogue SUID executables, and hidden processes (Tirumala et al., 2015). Tripwire, similarly, is a security and data integrity tool for monitoring and alerting on file and directory changes (G. H. Kim & Spafford, 1994). Other proposed HIDS systems that work with file system integrity monitoring are Disk-Based IDS (Griffin et al., 2003), Storage-Based IDS (Pennington et al., 2003) and I3FS (Patil et al., 2004);

- *System performance*: System performance parameters, such as central processing unit (CPU), random access memory (RAM), network utilisation, etc., can also be seen as a valuable source for intrusion and malware detection when exceeding the threshold barrier. For example, commercial HIDS Enterprise Immune System by Dark-Trace utilises artificial intelligence (AI) methods for detection of abnormal network flows. Moskovitch et al. proposed to use machine learning techniques using computer measurements, such as memory usage, disk usage, CPU usage, etc. (Moskovitch et al., 2007). They concluded that it is possible to detect previously un-encountered worms using the proposed approach, which is based on the computer “behaviour” (features). Vaas & Happa designed a client-server architecture that observes process’ memory consumption (Vaas & Happa, 2017). The reported results showed good performance of the system, which also allows for real-time monitoring.

Table 1.1. HIDS and NIDS comparison

IDS Location	NIDS	HIDS
Advantages	Attacks detected by checking network packets. No additional software is required to install on each host. Can check various hosts at the same time. Can view more global attack scope.	HIDS can check end-to-end encrypted communications behaviour. No extra hardware required. Detects intrusions by checking hosts file system, system calls or network events. Every packet is re-assembled. Looks at the entire item, not streams only.
Limitations	Challenge is to identify attacks from encrypted traffic. Dedicated hardware is required. It supports only the identification of network attacks. Difficult to analyse a high-speed network. The most serious threat is the insider attack.	Delays in reporting attacks. Consumes host resources. Needs to be installed on each host. It can monitor attacks only on the machine where it is installed.

End of the Table 1.1

IDS Location	NIDS	HIDS
Data source	Simple Network Management Protocol (SNMP). Network packets (TCP/UDP/ICMP). Management Information Base (MIB). Router NetFlow records.	System logs and system audit data. System calls. Windows registry. File system. System performance parameters.

As it can be seen from the description provided above, every type of IDS is specifically designed to work in its environment. The best result is achieved when both (or more) systems are used in the protection of the information system of the enterprise/company. A brief comparison of HIDS and NIDS can be seen in Table 1.1. HIDS and NIDS systems are both required because they provide significantly different benefits. Detection, deterrence, response, damage assessment, attack anticipation and prosecution support are available at different degrees from the different technologies. Table 1.2 summarises these differences.

Table 1.2. Comparing the benefits of NIDS and HIDS (Kozushko, 2003)

Benefit	HIDS	NIDS
Deterrence	Strong deterrence for insiders.	Strong deterrence for outsiders.
Detection	Strong insider detection. Weak outsider detection.	Strong outsider detection. Weak insider detection.
Response	Weak real-time response. Good for long-term attacks.	Strong response against outsider attacks.
Damage Assessment	Excellent for determining the extent of compromise.	Very weak damage assessment capabilities.
Attack Anticipation	Good at trending and detecting suspicious behaviour patterns.	None.
Prosecution Support	Strong prosecution support capabilities.	Very weak because there is no data source integrity.

1.2.2. Analysis Strategy of Detection

The first intrusion detection model was introduced in 1987 (Denning, 1987). It was presented as an IDES (intrusion-detection expert system) model that provided a sound basis for developing a powerful real-time intrusion detection capable of detecting a wide range of intrusions related to attempted break-ins, masquerading

(successful break-ins), system penetrations, Trojan horses, viruses, leakage and other abuses by legitimate users, and certain covert channels. Since then, the expert systems evolved, and intrusion detection methods (analysis strategies) can be broadly categorised into two main groups (Buczak & Guven, 2016):

- Signature-based (SIDS);
- Anomaly-based (AIDS).

SIDS is also known as Knowledge-based Detection or Misuse Detection and is designed to detect known attacks with only a small number of false-positives. It scans for pattern, that can be associated with known attacks against computer systems. Those patterns (provided by human experts) can be: hardware related parameters collection (CPU and RAM usage), the cryptographic hash value of rootkit or error log generated by an attack (Hay et al., 2008). Also, known systems vulnerabilities can be included in the signature database. Regular attacks patterns database update is required to have a fully functional system. This method is not effective against new (zero-day) and emerging cyber threats until they are added to the database. The increasing number of zero-day attacks is gradually rendering this this approach obsolete. Intruders can simply obfuscate their attack methods to bypass the predefined intrusion signatures. SIDS are employed in numerous common tools, for instance, Snort (Roesch, 1999) and NetSTAT (Vigna & Kemmerer, 1998). As SIDS is increasingly outperformed by AIDS, new proposals are still generated to increase its performance and tackle upcoming challenges. For example, one of the main problems is capability to extract signature information over multiple packets. This requires the IDS to remember the contents of earlier packets. Several studies related to SIDS proposed to use signatures created as state machines (Meiners et al., 2010), formal language string patterns or semantic conditions (P. C. Lin et al., 2011).

The idea of anomaly-based intrusion detection is predicated on a belief that an intruder's behaviour is noticeably different from that of a legitimate user and that many unauthorised actions are detectable. By design, this intrusion detection type is effective against zero-day attacks by raising an alert on new action that is not identified as a legitimate one (Chandola et al., 2009). Another advantage for this type – detection algorithm can be tailored for a specific company, network, or user, so it is very difficult for the attacker to select effective and non-detectable intrusion actions. Numerous machine learning methods of clustering and classification were applied for anomaly detection (Agrawal & Agrawal, 2015). The main disadvantage of that type of intrusion detection is the high false alarm rates (FAR). Large sets of training data are required to construct a normal behaviour profile, and it must be updated constantly. For example, if AIDS is relying on a system calls analysis, new system call traces are being generated along with an increasing number of new Windows and Linux applications. As AIDS only holds normal databases or data mining models of known behaviours, new normal system call

traces that do not conform to the databases or models may be falsely regarded as intrusions. It is also possible that malicious activity is included into the legitimate activity training data – in that case, the intrusive activity will be legit in later detection process. Therefore, it is extremely important to ensure the creation of “sterile” datasets that would include separately legitimate or malicious actions.

Table 1.3. Differences between signature-based detection and anomaly-based detection (H. Liu & Lang, 2019)

Feature	SIDS	AIDS
Detection performance	Low false alarm rate; High missed alarm rate.	Low missed alarm rate; High false alarm rate.
Detection efficiency	High, decrease with the scale of the signature database.	Dependent on model complexity.
Dependence on domain knowledge	Almost all detections depend on domain knowledge.	Low, only the feature design depends on domain knowledge.
Interpretation	Design based on domain knowledge, strong interpretative ability.	Outputs only detection results, weak interpretative ability.
Unknown attack detection	Only detects known attacks.	Detects known and unknown attacks.

The signature-based method has been used since the first implementation of intrusion detection systems and is now dominating the field. However, this type of intrusion detection was tailored for times when malware and cyber-attacks were spreading much slower than today and were more similar. Today, with existence of polymorphic malware and targeted attacks (which increases the number of zero-day attacks) – SIDS techniques progressively become less effective because no prior signature exists for such attacks (Symantec, 2019). Numerous different problems exist and are successfully resolved by applying AIDS solutions. For example, Vaas & Happa used ML method to detect a malware that is disguised as a legitimate program (Vaas & Happa, 2017). In another work, Chen et al. proposed to use Hidden Markov Model to detect planned network attacks (Chen et al., 2016). Conventional signature-based methods are failing at this particular task because of non-continuous attack profile. Canzanese et al. proposed how to use SVM method to detect malware that is new (zero-day attack) (Canzanese et al., 2015). A potential solution to aforementioned problems would be to adapt AIDS methods, which operates by profiling what is an ordinary behaviour rather than what is abnormal. All SIDS and AIDS differences can be seen in Table 1.3.

The following section is analysing anomaly-based methods used in intrusion detection. Furthermore, widely used AI and DL methods are introduced.

1.2.3. Events Prioritisation Problem in Intrusion and Malware Detection Systems

Two types of operations have been introduced to solve the problem of AV and IDS generated alerts classification (Alsubhi et al., 2008):

- *Low-level* (also called prioritisation) operations deal with each alert individually to enrich its attributes or assign a score to it based on potential risk.
- *High-level* (also called aggregation) alert management techniques, such as aggregation, clustering, correlation, and fusion, were proposed to deal with sets of alerts and provide their abstraction.

Both techniques have some disadvantages. High-level suffers from including alerts that are not significant. That leads to inappropriate results and increased load. The main problem with low-level techniques is that they have to automatically examine all alerts generated and prioritise them for further inspection, which also leads to improperly classified items.

In general, alert prioritisation operations use information from various domains that can be gathered from the network. Security policy, network topology, vulnerability analysis of network services and installed software, as well as asset profiles are the most used factors affecting the prioritisation of alerts (Chakir et al., 2018; Sadoddin & Ghorbani, 2006). The proposed systems use internal databases, which provide all the required information about monitored network topology. The security analyst can manually manage databases, or the system itself can automatically update them with the help of Nmap tool (Lyon, 2018).

Some attempts were made to automate malware classification by system calls (Kolosnjaji et al., 2016). However, authors do not motivate that task from the point of view of further attack risk prioritisation, and the size of malware dataset they used was twice smaller than in case of our experiments.

1.2.4. Techniques Used in Host-Based Intrusion and Malware Detection

Techniques used for the malware detection on the host machine can be broadly classified into three categories (similar to previously described IDS methods): malware signature, malware binary code, and dynamic malware behaviour. A dynamic behaviour technique is similar to the anomaly-based detection technique as it is using the knowledge of what is considered normal to find out what actually

is malicious. Signature-based detection uses the knowledge of what is considered as malicious to find out the maliciousness of the program under inspection.

- *Signature-based technique.* Commercial antivirus scanners look for signatures which are typically a sequence of bytes within the malware code to declare that the program scanned is malicious in nature and alert can be raised. Signatures can be used to identify particular viruses in executables, boot records, or memory with small error rate (Kephart & Arnold, 1994). However, the signature-based method is expensive and slow. In addition, this method is not effective against modified and unknown malicious executables. This disadvantage is the repetitive flaw of such method – intrusion and malware detection systems that are operating on signature-based manner cannot detect new and emerging attacks on information systems.
- *The binary code analysis technique.* This is quite a popular method and is often referred to as static analysis. Typically, API names and strings from malicious PE executables are extracted, and then this information is analysed with algorithms in order to find out the differences between malicious and benign programs (H. S. Anderson & Roth, 2018). Analysing malware by static analysis can achieve a very high detection speed, but the packing technique makes an automatic static analysis more difficult.
- *The dynamic behaviour technique.* It analyses the behaviour differences between malware and benign programs. This method needs to construct a secure environment for malware execution (also known as a SandBox). After execution, these behaviour records of each malware will be analysed to compare benign and malicious programs. Behaviour analysis can extract more features about malware than static analysis, but it takes more time than automatic static analysis because the static analysis does not need to execute malware (Norouzi et al., 2016). Behavioural malware detections have been proposed to overcome the limitations of static signature-based malware detection. Such detection captures the run-time behaviour of malware during its execution. Behavioural malware detection relies on various dynamic properties as features such as file system activities, terminal commands, network communications, and system calls (Alsulami et al., 2017). The detection function can be designed as in rule-based or learned such as in ML or DL algorithms. Machine learning techniques have been researched largely to build intelligent models that are capable of distinguishing between benign and malicious applications. Often ML or DL techniques have an advisory role in the IDS framework, and the human expert makes the final decision as a hybrid approach.

The main advantages and disadvantages of techniques used in host-based intrusion and malware detection can be seen in Table 1.4.

Table 1.4. Advantages and disadvantages of the techniques used in host-based intrusion and malware detection.

Technique	Advantages	Disadvantages
Signature-based	1. Small false positive rate. 2. Better detection rate in comparison with other techniques. Usually because this technique has good detection of known attacks. 3. Security personnel will save time on false positives analysis.	1. Technique is dependable on signatures database to perform at best detection rate. For such reason data-base must be updated frequently. 2. Does not detect new attacks or malware samples. For such reason Zero Day and evolving new attacks are not detected. 3. Signatures must be crafted and prepared before insertion to the database.
Binary code analysis	1. High detection speed. 2. Analyses binary without execution. It means that it can prevent possible damage.	1. Also does not detect Zero Day and emerging malware, that uses not known API calls configurations. 2. Packing technique prevents from successful data analysis.
Dynamic behaviour	1. Zero Day and emerging malware and intrusion attacks can be detected. 2. Installed system can learn normal behaviour from data generated by the information system (user behaviour). Or it can be easily extended by adding new samples of benign/malicious actions (malware samples).	1. False positives is the main problem of technique if training data is not extensive. 2. If any malicious actions are performed when normal behaviour learning is executed – trained model will be compromised.

1.3. Anomaly-based Intrusion and Malware Detection

Anomaly detection is described as a process of finding the patterns in a dataset whose behaviour is not normal or expected. These unexpected behaviours are also termed as anomalies or outliers (Agrawal & Agrawal, 2015). Outlier detection has been widely researched not only from the data mining perspective, but also in statistics and machine learning. This task at first glance may appear simple, but

that is not true. In most cases, data elements are called outliers only in a specific type of context.

Outlier detection in literature is also represented as anomaly detection, deviation detection, novelty detection, exception mining, etc. One important issue of outlier detection is the nature of data. In outlier detection, perspective data is also regarded as a point, object, record, vector, sample, observation, etc.

For outlier detection, there are two general data types: simple and complex. Simple data is single typed real-valued low dimensionality attributes. Complex: high dimensionality and different type (streaming data, sequence data).

In literature, a classification of outliers also exists. It depends on the provided data and classification methods used to detect anomalies (Pathan, 2014):

- *Point anomalies*. When a particular data instance does not follow the rest of the data, it can be assigned to the outlier class. As a realistic example, consider car fuel expenditure. If usual fuel consumption a day is 5 litres and one day it becomes 50 litres, then this is obviously a point outlier;
- *Contextual anomalies*. When a data instance is behaving anomalously in a particular context and normally otherwise, then it is termed as a contextual outlier, which can also be referred to as a conditional outlier. For example, the expenditure on a credit card during a festive period is not like at a regular time of year. If the regular expenditure every month is \$500, which is normal, then an expenditure of \$2000 during a non-festive period is considered abnormal and hence is a contextual outlier;

Collective anomalies. When a collection of similar data instances is behaving anomalously, then with respect to the entire data set that collection is termed a collective outlier. It might be the case that the individual data instance is not an outlier by itself, but due to its presence in a collection, it is becoming an outlier (Chandola et al., 2009).

Anomaly detection methods can be categorised into three types (Raut & Singh, 2014):

- *Supervised*. These approaches require the learning of normal and outlying models from the knowledge of prelabelled data. According to this learning, supervised techniques detect a new data instance as normal or outlying. It is tested with the normal and outlying models and checks where it is fitted. Supervised approaches are basically used in intrusion detection, however gaining access to prelabelled data is the main challenge for these approaches;
- *Unsupervised*. These approaches do not require any prelabelled data to detect outliers. For example, in normally distributed data, we use the three-sigma rule to detect outliers. Some methodologies use distance to detect outliers without any prelabelled data. Compared with supervised

approaches, unsupervised approaches are more applicable in various real-life domains;

- *Semi supervised*. These approaches require prelabelled data but only normal class. These approaches do not require the outlying class data to be labelled beforehand, thus these approaches can be widely applicable and more usable than supervised techniques.

Anomaly detection is popular in the following areas:

- *Intrusion detection*. Computer intrusion includes hacking and spreading of viruses and worms across networks to infiltrate a local or remote machine or cause damage using distributed denial of service (DDoS) attacks (Raut & Singh, 2014);
- *Fraud detection*. Fraud detection refers to detection of criminal activities occurring in commercial organisations such as banks, credit card companies, insurance agencies, cell phone companies, stock market, and so on;
- *Medical and public health anomaly detection*. Anomalous records can be generated due to patient condition or instrumental error or recording errors. A wrong test report might have serious repercussions. On the other hand, outlier detection in this domain is a very important tool that can potentially save human lives by detecting a problem quickly from test results and images (Pathan, 2014);
- *Industrial damage detection*. Industrial units suffer damage due to continuous usage and normal wear and tear. Such damage needs to be detected early to prevent further escalation and losses. The data in this domain is usually referred to as sensor data because it is recorded using different sensors and collected for analysis. Anomaly detection techniques have been extensively applied in this domain to detect such damage;
- *Image processing*. Anomaly detection techniques dealing with images are either interested in any changes in an image over time (motion detection) or in regions that appear abnormal on the static image;
- *Anomaly detection in text data*. Anomaly detection techniques in this domain primarily detect novel topics or events or news stories in a collection of documents or news articles. The anomalies are caused due to an interesting new event or an anomalous subject;
- *Sensor networks*. A sensor network might comprise of a multiple number of sensors in which each sensor collects different types of data. The collected data often contains noise and missing values due to limitations created by environmental and communication channels. Outlier detection in a sensor network has several real-life applications, such as environmental

monitoring, habitat monitoring, industrial monitoring, target tracking, surveillance monitoring, etc. (Chandola et al., 2009).

Intrusion detection is one of the domains where anomaly-based methods are successfully applied. One of the earliest works that proposed intrusion detection by identifying abnormal behaviour can be attributed to Anderson (J. P. Anderson, 1980). In his report, Anderson presents a threat model that classifies threats as external penetrations, internal penetrations, and misfeasance, and uses this classification to develop a security monitoring surveillance system based on detecting anomalies in user behaviour. External penetrations are defined as intrusions that are carried out by unauthorised computer system users; internal penetrations are those that are carried out by authorised users who are not authorised for the data that is compromised, and misfeasance is defined as the misuse of authorised access both to the system and to its data. In a similar paper, Denning put forth the idea that intrusions to computers and networks could be detected by assuming that users of a computer/network would behave in a manner that enables automatic profiling (Denning, 1987). In other words, a model of the behaviour of the entity being monitored could be constructed by an intrusion detection system, and subsequent behaviour of the entity could be verified against the entity's model. The model includes profiles for representing the behaviour of subjects with respect to objects in terms of metrics and statistical models, and rules for acquiring knowledge about this behaviour from audit records and for detecting anomalous behaviour. In this model, behaviour that deviates sufficiently from the norm is considered anomalous. In the paper, Denning mentioned several statistical models that can be used on anomaly-based IDS: Operational model, Mean and standard deviation model, Multivariate model, Markov process model and Time Series model.

The main premise of anomaly detection is that invasive activity is a subset of anomalous activity. If we consider an outsider, who has no idea of a legitimate user's activity patterns, intruding into a host system, there is a strong probability that the intruder's activity will be detected as anomalous. However, intrusive activity does not always coincide with anomalous activity. The four possibilities have been suggested, each with a non-zero probability (Patcha & Park, 2007):

- *Intrusive but not anomalous*: These are false negatives. An intrusion detection system fails to detect this type of activity as the activity is not anomalous. These are called false negatives because the intrusion detection system falsely reports the absence of intrusions;
- *Not intrusive but anomalous*: These are false positives. In other words, the activity is not intrusive, but because it is anomalous, an intrusion detection system reports it as intrusive. These are called false positives because an intrusion detection system falsely reports intrusions;

- *Not intrusive and not anomalous*: These are true negatives; the activity is not intrusive and is not reported as intrusive;
- *Intrusive and anomalous*: These are true positives; the activity is intrusive and is reported as such.

The most effective IDS have a small number of false negatives. That can be achieved by setting thresholds (that define anomaly) to lower values. This action is often resulted in many false positives and reduces the effectiveness of automated intrusion detection. As a result of imbalanced false negatives and false positives, the security personnel are loaded with additional work. Every alert produced by IDS must be reviewed comprehensively. Many approaches have been proposed to balance false negatives and false positives in automated IDS. One of the first methods has been based on knowledge and statistical techniques. Later, with the evolution of computer systems, more advanced machine learning methods emerged (such as Neural networks or Support Vector machines). The increase of processing power in recent years has led to deep learning methods application in intrusion detection. The DL methods not only introduce a better detection rate than typical ML algorithms, but they are also capable of working with an enormous amount of data (also known as the Big Data). All mentioned methods will be discussed in the next subsection.

1.3.1. Anomaly-Based Techniques of Malware and Intrusion Detection

Anomaly-based methods for IDS are categorised into three main groups: Statistics-based, knowledge-based, machine learning-based (Buczak & Guven, 2016; Elhag et al., 2015; Xin et al., 2018). The ML group has new emerging subgroup – DL.

- A *statistics-based* IDS builds a distribution model for normal behaviour profile, then detects low probability events and flags them as potential intrusions. Statistical IDS essentially take into account the statistical metrics such as the median, mean, mode and standard deviation of analysed data (internet packets, user actions, PC parameters or system calls). Statistical IDS are employed to identify any type of differences in the present behaviour from normal behaviour. A most common example of a statistics-based IDS is HIDE (Z. Zhang et al., 2001). HIDE is an anomaly-based network intrusion detection system that uses statistical models and neural network classifiers to detect intrusions;
- A *knowledge-based* IDS uses information entered by the experts of the field to detect an anomaly. For that reason, these systems are often called expert-based IDSs. This approach requires creating a knowledge base

which indicates the normal traffic profile. Actions which differ from this standard profile are labelled as an intrusion. Unlike the other classes of IDS, the standard profile model is normally created based on human (expert) knowledge, consisting of a collection of rules that try to define normal system activity. One of the main advantages of knowledge-based IDS is a reduced number of false-positives since the system has knowledge of all the normal behaviours. However, the update of rules is a time and labour intensive task. One of the first knowledge-based IDS is Haystack (Smaha, 1988). It was developed to detect intrusion by analysing user log profiles. Signature analysis is another knowledge-based technique – it searches for already known fragments of information in the analysed data (internet packets, log files, etc.) (Berlin et al., 2015);

- *Machine learning* is the process of extracting knowledge from large quantities of data. Generally, machine learning is considered to be the process of applying a computing-based resource to implement learning algorithms. Formally, machine learning has been defined as the complex computation process of automatic pattern recognition and intelligent decision-making based on training sample data (Dua & Du, 2016). Lately, ML usage in IDS is the most popular technique. One of the advantages of ML is that it does not need to have deep knowledge of data analysed: ML methods are capable of extracting required features (or even minimise feature vector size to the minimal size) from provided datasets (Sung & Mukkamala, 2003). Furthermore, various combinations of such methods allow researchers to achieve high accuracy and low FPR results in IDS area (Mora-Gimeno et al., 2021).

Table 1.5. Advantages and disadvantages of the anomaly-based techniques of malware and intrusion detection

Technique	Advantages	Disadvantages
Statistics-based	1. None of the prior knowledge about the signatures of the attacks is required. 2. Easy to maintain, because no signatures are required.	1. Accurate statistical distribution is required. 2. Long learning process. 3. Cannot handle high dimension data.
Knowledge-based	1. Produces good accuracy. 2. Produces low false positive rates.	1. It is required to collect a prior knowledge. This task is time consuming and difficult/impossible to automate. 2. Does not detect Zero-Day or emerging attacks.

End of the Table 1.5

Technique	Advantages	Disadvantages
Machine learning-based	1. Can handle high dimensional data and extract important features. 2. Does not need to have a deep knowledge about analysed data.	1. False positives is the main problem of technique if training data is not extensive. 2. Long training time. 3. High algorithmic complexity as the model is seen as black box and often cannot explain why certain results are returned.

The main advantages and disadvantages of anomaly-based techniques of malware and intrusion detection can be seen in Table 1.5.

1.3.2. Application of Machine Learning Methods in Anomaly-based Intrusion and Malware Detection

There are many misunderstandings about the connection between ML, DL, and artificial intelligence (AI). AI is a new technological science that studies and develops theories, methods, techniques, and applications that simulate, expand and extend human intelligence (Smith & Eckroth, 2017). It is a branch of computer science that aims to understand the essence of intelligence and to produce a new type of intelligent machine that responds in a manner similar to human intelligence. Research in this area includes robotics, computer vision, natural language processing and expert systems. AI can simulate the information process of human thinking and consciousness.

ML is a branch of AI and is closely related to computational statistics, which also focuses on using computers in making predictions. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. ML is occasionally conflated with data mining, but the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning (Louridas & Ebert, 2016). ML can also be unsupervised and can be used to learn and determine baseline behavioural profiles for various entities and then used to find meaningful anomalies (Jordan & Mitchell, 2015). ML primarily focuses on classification and regression based on known features learnt from the earlier training data.

DL is a new field in machine-learning research. Its stimulus lies in the foundation of a neural network that imitates the human brain for analytical learning. It mimics the human brain mechanism to interpret data such as images, sounds and texts to allow computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction (LeCun

et al., 2015). The concept of DL was proposed by Hinton based on the deep belief network (DBN), in which an unsupervised greedy layer-by-layer training algorithm is proposed that provides hope for solving the optimization problem of deep structure (Hinton, 2009). Then the deep structure of a multi-layer automatic encoder is proposed. In addition, the convolution neural network proposed by LeCun et al. is the first real multi-layer structure learning algorithm that uses a space relative relationship to reduce the number of parameters to improve the training performance (Lecun et al., 1998). The result of that proposal - convolutional NN's have been applied to eliminate the need for hand-crafted feature extractors (Deng & Yu, 2013). In this thesis – ML methods are called simple methods and DL – advanced.

It is well known that attackers often use various techniques to transform and hide malicious activity from the signature-based IDS. Anomaly-based techniques have been used to fight such problems and introduced a better detection rate of unknown attacks but increased the number of false-positives because of primitive approaches applied. Advanced deep learning methods in combination with extensive training datasets are required to build a benign behaviour profile and decrease the FAR rate.

Several ML classification and clustering methods have been popular among researchers to improve anomaly-based IDS detection rate (listed in no particular order): Support Vector Machines, Naïve Bayes, K-Means Clustering, Genetic algorithms, Fuzzy logic, Hidden Markov Model, K-Nearest Neighbors, Decision trees and Neural Networks.

Support Vector Machines or *SVMs*, are learning machines that plot the training vectors in high-dimensional feature space, labelling each vector by its class. SVMs classify data by determining a set of support vectors, which are members of the set of training inputs that outline a hyperplane in the feature space (Sung & Mukkamala, 2003). SVM without a kernel is a single neural network neuron but with a different cost function. If you add a kernel function, then it is comparable with 2-layer neural nets. The first layer is able to project data into some other space, and the next layer classifies the projected data. If one more layer is added, then multiple kernel SVMs is ensembled and 3-layer neural network is mimicked.

Naïve Bayes approach is based on applying Bayes' principle with robust independence assumptions among the attributes (Wu et al., 2008). Naïve Bayes answers questions such as “what is the probability that a particular kind of attack is occurring, given the observed system activities?” by applying conditional probability formulae. Naïve Bayes relies on the features that have different probabilities of occurring in attacks and in normal behaviour. This method is producing good detection results with small datasets. But most importantly – it requires a minimal amount of calculations to produce results. However, large datasets are not suitable

for this method – accuracy is reduced, and the system does not operate as intended when complex attribute dependencies are presented (Koc et al., 2012).

K-Means Clustering. Clustering is the division of data into groups of similar objects. In clustering, some details are disregarded in exchange for data simplification. Clustering can be viewed as a data modelling technique that provides for concise summaries of the data. Clustering is therefore related to many disciplines and plays an important role in a broad range of applications (Berkhin, 2006).

Genetic algorithms are a heuristic approach to optimization, based on the principles of evolution. Each possible solution is represented as a series of bits (genes) or chromosome, and the quality of solutions improves over time by the application of selection and reproduction operators, biased to favour fitter solutions (Goldberg & Holland, 1988). For example, GA has been used to evolve simple rules for network traffic. Every rule is represented by a genome, and the primary population of genomes is a number of random rules. Each genome is comprised of different genes which correspond to characteristics such as IP source, IP destination, port source, port destination and protocol type (Sazzadul Hoque, 2012).

The point of *fuzzy logic* is to map an input space to an output space, and the primary mechanism for doing this is a list of if-then statements called rules. All rules are evaluated in parallel, and the order of the rules is unimportant. The rules themselves are useful because they refer to variables and the adjectives that describe those variables. Therefore, it presents a straightforward way of arriving at a final conclusion based upon unclear, ambiguous, noisy, inaccurate or missing input data. For this reason, fuzzy logic is a good classifier for IDS problems as the security itself includes vagueness, and the border line between the normal and abnormal states is not well identified (Sekeh & Maarof, 2009). In addition, the intrusion detection problem contains various numeric features in the collected data and several derived statistical metrics. One of the examples of fuzzy logic usage in IDS is work done by Elhag et al. They showed that with fuzzy logic the false alarm rate in determining intrusive actions could be decreased. They outlined a group of fuzzy rules to describe the normal and abnormal activities in a computer system, and a fuzzy inference engine to define intrusions (Elhag et al., 2015).

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unseen data. Prior research has shown that HMM analysis can be applied to identify particular kinds of malware (Annachhatre et al., 2015). In this method, a Hidden Markov Model is trained against known malware features (e.g., operation code sequence) and once the training phase is completed, the trained model is applied to score the incoming network traffic. The score is then contrasted to a predefined threshold, and a score greater than the threshold indicates malware. Likewise, if the score is less than the threshold, the network traffic is classified as normal.

Decision trees are a supervised ML algorithm that is known to perform well if it is well configured. It supports both numerical and categorical data, and it can handle large data sets. A Decision Tree can be represented with nodes and edges, and it is composed of a root node that performs the first split and leaf nodes in which we can find the predicted results. Generally, splitting is based on entropy and information gain (Hafsa & Jemili, 2018).

Neural Networks. The artificial neural network algorithm studied as part of machine learning is a statistical learning algorithm that developed logically from biological neural networks. The very first paper on neural networks was produced in 1943: Professors McCulloch and Pitts published a paper titled "A Logical Calculus of the Ideas Immanent in Nervous Activity," which logically explained the human neural network and conceptualized the ANN for the first time in history (Jin Kim et al., 2017). Artificial neuron models are at their core simplified models based on biological neurons. This allows them to capture the essence of how a biological neuron is functioning. Usually, artificial neurons are called perceptrons. An artificial neural network consists of a group of processing elements (perceptrons) that are greatly interconnected and convert a set of inputs to a set of preferred outputs. The result of the transformation is determined by the characteristics of the elements and the weights associated with the interconnections among them. By modifying the connections between the nodes, the network is able to adapt to the desired outputs (Iftikhar Ahmad et al., 2009).

A very important property of neural networks is the concept of programming by example. A large number of weights makes it difficult to fix them to obtain the desired result. Instead, the net is programmed by example and repetition. It is trained by presenting input-output pairs repeatedly. Each time an input is presented, the network guesses an output. The output part of the input-output pair is used to determine whether the network is right or wrong. When wrong, the network is corrected by a learning algorithm using a gradient method on the output error to modify the weights. After each modification, the network gets closer to the desired transfer function as represented by the sample.

The ANN consists of a series of layers; an input layer, an output layer, and one or more hidden layers. Each layer is made up of multiple nodes (i.e., artificial neurons) operating in parallel (Chaibi et al., 2020). In ANNs, artificial neurons in consecutive layers are connected through a series of links which have assigned adaptive weights (which are calibrated during the training process using back-propagation algorithms). The training of ANN typically consists of two stages; feedforward (or activation propagation) and error backpropagation. The tangent sigmoid and the logarithmic sigmoid are common activation functions used in ANN for neurons in the hidden layer.

At the end of the feedforward stage, the final output vector is compared to a target output; commonly through the calculation of the mean squared error (MSE).

The error is then backpropagated from the output layer to the input layer using a backpropagation training algorithm to adjust weights of the connecting links for minimization of the MSE. The error backpropagation stage continues until a convergence criteria is reached.

Table 1.6. Advantages and disadvantages of the most popular ML methods

Method	Advantages	Disadvantages
Decision Tree	1. There are no requirements of domain knowledge in the construction of the decision tree. 2. It minimizes the ambiguity of complicated decisions and assigns exact values to outcomes of various actions. 3. It can efficiently process the data with a high dimension. 4. It is easy to interpret. 5. Decision tree also handles both numerical and categorical data.	1. It is restricted to one output attribute. 2. It generates categorical output. 3. It is an unstable classifier, i.e. performance of the classifier depends upon the type of dataset. 4. If the type of dataset is numeric then it generates a complex decision tree
Support Vector Machine	1. Better Accuracy as compared to the other classifiers. 2. Easily handle complex nonlinear data points. 3. Overfitting problem is not as much as other methods.	1. Computationally expensive. 2. The main problem is the selection of the right kernel function. For every dataset different kernel function shows different results. 3. As compared to other methods training process take more time. 4. SVM was designed to solve the problem of binary class. It solves the problem of the multi-class by breaking it into a pair of two classes such as one-against-one and one-against-all.
Neural Network	1. Easily identify complex relationships between dependent and independent variables. 2. Able to handle noisy data.	1. Local minima. 2. Over-fitting. 3. The processing of ANN network is challenging to interpret and requires high processing time if there are large neural networks.
Bayesian Belief Networks	1. It makes computations process easier. 2. Have better speed and accuracy for massive datasets.	1. It does not give accurate results in some cases where dependency among variables exists.

However, the most significant disadvantage of applying Neural Networks to intrusion detection is the “black box” nature of the Neural Network. The “Black Box Problem” has overwhelmed Neural Networks in a number of applications (Iftikhar Ahmad et al., 2009). This Black Box neural networks feature is preventing researchers from clearly seeing and analysing learning results. It is hardening network tinkering process as researchers cannot clearly analyse and modify network for better outcomes.

After reviewing ML methods, it is essential to have combined advantages and disadvantages of the most popular methods. Those features are provided in Table 1.6.

Since 2006, the deep-structured learning, more commonly known as deep learning or hierarchical learning, has emerged as a new area of machine learning research (Deng & Yu, 2013). The architecture of Deep Neural Network is based on many layers of Neural Networks (NN). DNN and new hardware capabilities revived the current state of ANN research. Two robust DNN designs were introduced as a result: Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

CNN's, or ConvNet, are mainly applied for the image recognition tasks because they can scale adequately on large images and are designed to mimic the human visual system. The CNN consists of convolutional layers, subsampling (pooling) layers, and fully connected layers. The common CNN architecture is primarily a combination of convolutional layers and subsampling layers. The fully connected layers are the upper layers, the input is the features extracted by the lower layers, and the output is the classification result. The first CNN model was introduced by Lecun et al in 1998. The model has been named Le-Net-5 and was used for document analysis (Lecun et al., 1998). The usage of convolution and pooling layers allowed to reduce the size of feature maps when large data were managed (e.g. pictures) (Dumoulin & Visin, 2016; Lecun et al., 1998). The sample of CNN architecture for static malware classification with one convolutional layer is illustrated in Figure 1.3.

Convolutional layers are the most important part of CNN. A convolutional layer consists of a set of learnable kernels which are convolved across the width and the height of the input features during the forward pass producing a 2-dimensional activation map of the kernel. To sum up, a kernel consists of a layer of connection weights with the input being the size of a small 2D patch and the output – a single unit.

The outputs of a linear operation such as convolution are then passed through a nonlinear activation function. Although smooth nonlinear functions, such as *sigmoid* or *hyperbolic tangent* (tanh) function, were used previously because they are mathematical representations of a biological neuron behaviour, the most common

nonlinear activation function used presently is the *rectified linear unit* (ReLU), which simply computes the function: $f(x) = \max(0, x)$ (Yamashita et al., 2018).

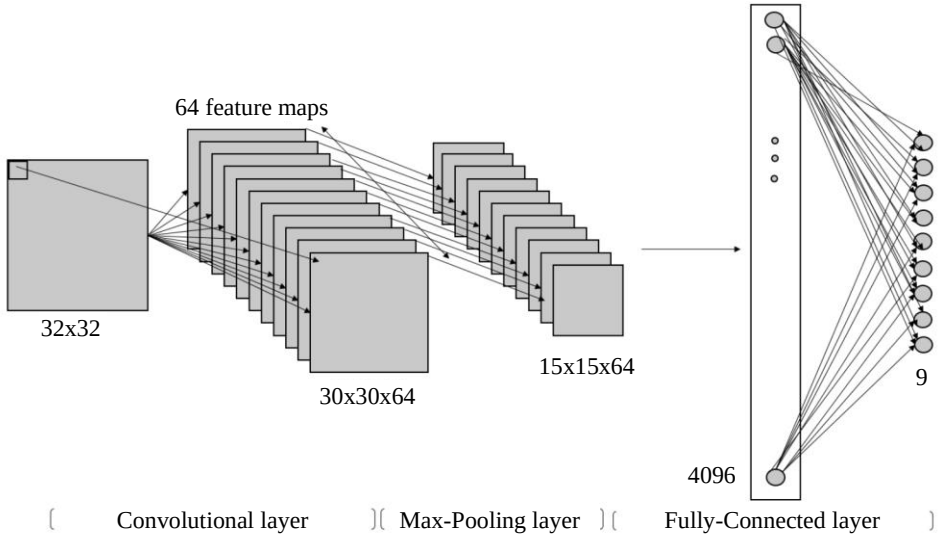


Fig. 1.3. Fully Convolutional Neural Network architecture (Gibert, 2016)

Pooling is a form of non-linear down-sampling. There are several non-linear functions to implement pooling such as the minimum, the maximum and the average, but the most common is the maximum. Max pooling works by partitioning the image into a set of non-overlapping rectangles and for each sub-region outputs the maximum value.

In general, all CNN's models used adopt the following four basic steps to achieve its results: convolve several small filters on the input image, subsample this space of filter activations (pooling step), repeat steps 1 and 2 until you are left with enough high-level features, apply a common feed-forward NN to the extracted features.

The opinion that deeper CNN's are much more successful than shallower architectures is highly common and defended in the literature (He et al., 2016). In fact, in the space of 4 years, winning CNN configuration of ImageNet competition (Jia Deng et al., 2009) went from 7 layers in AlexNet 2012 (Krizhevsky et al., 2012) to 1000 layers for ResNet 2016 (He et al., 2016). These types of deep architectures generally need a huge amount of data in order to generalize well on unseen examples (He et al., 2016). Furthermore, the generic computer resources

cannot handle the new complex CNN models. For example, one of the latest winners of ImageNet used 32 GPUs (NVIDIA Tesla-V100) to train EfficientNet-B0/B4, and 256 GPUs for EfficientNet-B7/B8 models (Wei et al., 2020). They used the competition subset which has 1K classes, 1.3M training and 50K validation images.

Today, CNNs are deployed in many practical applications in the fields of computer vision and natural language processing. CNNs were used by the winners of several competitions, such as aforementioned ImageNet, Kaggle Facial Expression, Kaggle Multimodal Learning, Kaggle Canadian Institute for Advanced Research (CIFAR)-10, and German Traffic Signs (Graham, 2014; Russakovsky et al., 2015; Stallkamp et al., 2011).

Another branch of DL - RNN is capable of working with time series-based data. RNNs are a form of network with backward connections, where the output from a layer in the network is fed back into either that layer or a previous layer in the network. RNNs maintain state, in that values calculated at a previous time step are used in the current timestep. This state is used as a form of short-term memory in the network, and RNNs are frequently used as a model for time series and sequential data, where values at previous time steps can affect the current calculation. Usually, this definition fits perfectly for speech recognition and language translation tasks. A typical RNN architecture was used by Le et al. to verify proposed sequence forward selection algorithm and decision tree model for network-level intrusion detection (Le et al., 2019).

The training of RNN is similar to the training of the traditional neural network. This model also uses a backpropagation algorithm but with a slight difference. Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time steps but also the previous time steps. This is called backpropagation through time (BPTT) (Deng & Yu, 2013). Unfortunately, vanilla RNNs trained with BPTT have difficulty learning long-term dependencies because of the so-called vanishing/exploding gradient problem.

The long short-term memory (LSTM) networks were introduced in 1997 as a solution for the vanishing/exploding gradient of RNN (Hochreiter & Schmidhuber, 1997). A simple idea has been proposed: augmenting the network with an explicit memory, and was implemented using special hidden units, the natural behaviour of which is to remember inputs for a long time. A special unit called the memory cell acts like an accumulator or a gated leaky neuron: it has a connection to itself at the next time step that has a weight of one, so it copies its own real-valued state and accumulates the external signal, but this self-connection is multiplicatively gated by another unit that learns to decide when to clear the content of the memory (Hochreiter & Schmidhuber, 1997).

The most popular LSTM usage is to combine it with CNN in one network as one of the last layers. For example, LSTM have been added before the last decision-making layer in proposed CNN-LSTM architecture, named BD-LSTM. It was used to detect anomalies in video surveillance data (Ullah et al., 2020).

In 2014 another, simpler, RNN model has been proposed – gated recurrent unit (GRU). This model combines the forget gate and input gate into a single update gate. It also merges the cell state and hidden state and makes some other changes. The resulting model is simpler than standard LSTM models and has been growing increasingly popular since presentation (Cho et al., 2014). Due to fewer amount of gates, GRU is faster to train (Chung et al., 2014). A representative example of GRU usage in DL network can be considered a research where the proposed model was used for the network-based intrusion detection (Xu et al., 2018).

Finally, it can be stated that RNNs are dynamic systems; they have an internal state at each time step of the classification. This is due to circular connections between higher- and lower-layer neurons and optional self-feedback connections. These feedback connections enable RNNs to propagate data from earlier events to current processing steps (Lipton et al., 2015). The natural capability of RNNs to accept and work with sequences has allowed to show outstanding performance in speech recognition and machine translation (Bahdanau et al., 2015).

Table 1.7. Advantages and disadvantages of the most popular ML methods

Method	Advantages	Disadvantages
RNN	1. Approximate non-linear functions. 2. Robust to noise. 3. Can handle multivariate inputs. 4. Temporal dependence.	1. The processing of RNN network is challenging to interpret and require high processing time. 2. Gradient vanishing and exploding problems. 3. It cannot process exceptionally long sequences if using <i>tanh</i> or <i>relu</i> as an activation function (usually those functions returns the highest results).
CNN	1. Approximate non-linear functions. 2. Robust to noise. 3. Can handle multivariate inputs. 4. Feature learning possibilities.	1. Cannot learn temporal dependence. 2. The processing of RNN network is challenging to interpret and require high processing time. 3. Lots of training data is required.

Finally, advantages and disadvantages of RNN and CNN DL methods are provided in Table 1.7. It is worth repeating that RNN and CNN models have a common advantage with NN – they yield high accuracy results but are “black box” when the analysis is required, and this falls to the common disadvantage.

Furthermore, the RNN and CNN methods were used for further DL experiments as they were determined by a number of authors as the most accurate and they tend to generate better FPR results than most popular ML methods (NN, SVM or HMM) (Al Jallad et al., 2020; Bouzar-Benlabiod et al., 2020; Meng et al., 2017).

1.3.3. Datasets Designed for Malware and Intrusion Detection

Most of the recent research on intrusion detection has been done using a network-based DARPA and KDD Cup 99 datasets. So far, 42 % KDD cup dataset, 20% DARPA dataset and 38 % other datasets have been used to verify proposed new methods for anomaly detection (Azad & Jha, 2013; Srivastav et al., 2020). Furthermore, the research done in the sphere of anomaly dataset generation for HIDS training is minimal despite the growing need for anomaly-based HIDS systems. KDD Cup 99 dataset was collected in 1999 by processing the tcpdump portions of the 1998 DARPA Intrusion Detection System (IDS) Evaluation dataset, created by Lincoln Lab under contract to DARPA (Brugger, 2007; R.P. Lippmann et al., 1999). Those datasets contain various information collected on simulating attacks against a network. Four main attack types have been used against a simulated US Air Force LAN:

- *Probing*. Probing is a class of attacks where an attacker scans a network to gather information or find known vulnerabilities. An attacker can find related exploits if network machine's map with corresponding services is available;
- *Denial of service attacks*. DoS is a class of attacks where an attacker makes some computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate users access to a machine;
- *User to Root attacks*. The attacker starts on a local normal user account, and after some commands and related exploits usage – gains root user account control;
- *Remote to User attacks*. On this type of the attack, the attacker sends commands to the target machine (one uses already known exploits for that machine) and illegally gain local access as a user;

Therefore, DARPA-related datasets have data associated with a network and are perfect for applying in NIDS research (Sahu et al., 2014). It is necessary to stress that DARPA, which has been used as a de facto standard for anomaly-based NIDS training, present the simulated and not the real attack data which is more valuable in the field of intrusion detection (M. Liu et al., 2019). Nevertheless, it is still considered by experts as a valuable dataset.

It is worth to mention that some host-based information has been collected during the KDD dataset assembly. At first, it was a Unix-type host systems information. Later, in January 2000, Windows NT hosts data was collected on similar circumstances (Korba, 2000). It contains not only tcpdump provided data, but also the Windows NT event log audit data. Despite the provided collections, KDD Cup-related datasets lack host machines-related information and only NIDS researches use it (Waqas Haider et al., 2016).

To improve the situation mentioned above, researchers proposed several publicly available datasets with the inclusion of system calls from newer operating systems and with more features. Unfortunately, the most comprehensive datasets have been collected from Linux-based operating systems: FirefoxDS (Murtaza et al., 2013), ADFA-LD (Creech & Hu, 2013) and NGIDS-DS (W. Haider et al., 2017):

- Firefox-DS has been created using modern penetration testing techniques such as Metasploit. The dataset contains normal and anomalous system call traces from multiple programs. In Firefox-DS, normal traces are acquired by running standard executions of the Firefox Internet browser. Five up-to-date attacks such as memory corruption exploit, integer overflow attack, DOM exploit, and pointer exploit are launched against Firefox 3.5, and complete system call traces are collected. Traces are grouped according to distinct Firefox processes, and each trace contains massive system calls (Murtaza et al., 2013);
- The ADFA-LD was created on the Linux operating system with several vulnerabilities. The system was attacked by penetration testing tools with several contemporary attack methods. System calls numbers have been used as the main data of dataset;
- Haider et al. proposed a synthetical high-quality and realistic IDS dataset named the next-generation IDS dataset (NGIDS-DS) using the advantageous Ixia Perfect Storm commercial hardware platform and infrastructure in ADFA (W. Haider et al., 2017). The dataset contains 99 CSV files of labelled host logs for the design of HIDS as well as network packets for the design of NIDS. The host logs contain various information, such as the time of activities, process IDs, execution paths, and system call numbers for comprehensive system call analysis. They also assessed the quality of existing IDS datasets including DARPA and ADFA-LD with a fuzzy logic system based on the “Sugeno fuzzy inference model”.

Moreover, several attempts have been made to generate novel public datasets for the Windows operating system – in 2015 database of Windows audit logs were generated by running malware samples, and new malicious behaviour detection methods were introduced (Berlin et al., 2015). They demonstrated that audit logs

could potentially provide an effective, low-cost signal for detecting malicious behaviour on enterprise networks. Furthermore, proposed logistic regression machine-learning classifier generated a detection rate of 85% at an expected false positive rate of 1%.

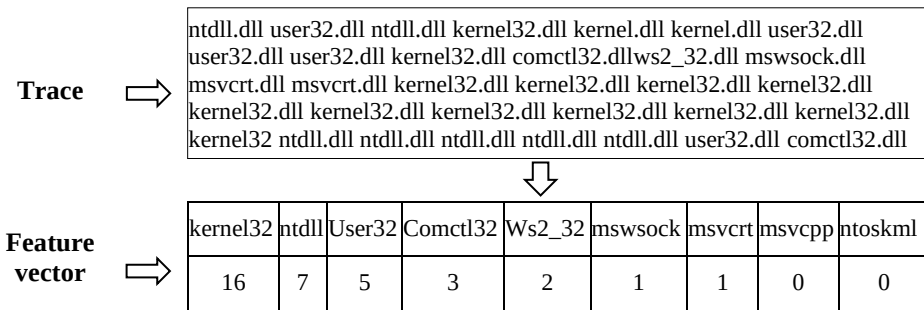


Fig. 1.4. ADFA-WA feature construction scheme (Waqas Haider et al., 2016)

In 2016 ADFA-IDS dataset, containing Windows system calls generated in the experiment when zero-day attacks were simulated, was introduced (Creech & Hu, 2014; Waqas Haider et al., 2016). It is rather widely used, but even the authors of ADFA-IDS agree that ADFA Windows datasets (ADFA-WA) (Figure 1.4) are incomplete: only basic information was collected (Table 1.8), and an insufficient amount of vulnerabilities was used to generate malicious activity (Waqas Haider et al., 2016). Despite these disadvantages ADFA family datasets are widely used by researchers for various DL and statistical methods verification (Z. Liu et al., 2020).

Table 1.8. ADFA-WA dataset characteristics in the mean of collected data count

Dataset	Normal Training	Normal Validation	Attack
ADFA-WD	356 traces	1828 traces	5773 traces
ADFA-WD:SAA	same as above		863 traces

Recently, scientists at the Canadian Institute of Cybersecurity have introduced the CSE-CIC-IDS2018 dataset, which includes the network traffic and log files of Windows and Linux machines (Sharafaldin et al., 2018). Still, the main attention is paid to the network level attacks such as Brute-force, Heartbleed, Botnet, DoS, DDoS, Web attacks, and infiltration of the network from inside. The authors of CIC-IDS2018 have concluded that common Random Forest ML method is the fastest and produces a comparable precision result of 0.98. Initial

experiments have been executed with network-based data, and newly introduced Windows data was not used.

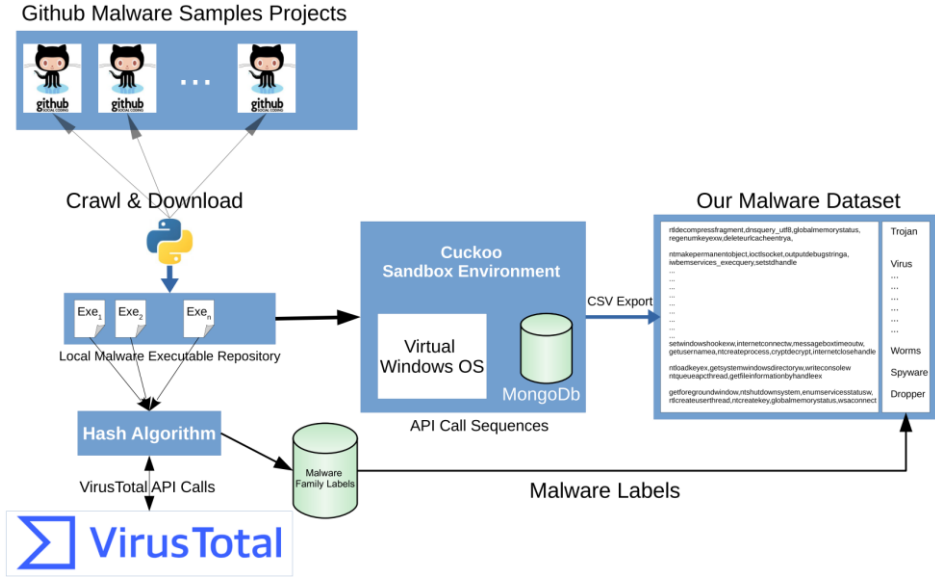


Fig. 1.5. Mal-API-2019 dataset generation scheme (Catak & Yazı, 2019)

All previously mentioned datasets belong to the IDS field. But there also exists publicly available datasets that are oriented to the antivirus applications. One of the latest is Mal-API-2019 (Catak & Yazı, 2019). The main objective of the aforementioned study was to obtain data which would help to address machine learning-based malware research gaps. The specific objective of this study was to build a benchmark dataset for Windows operating system API calls of various malware. This was the first study to undertake metamorphic malware to build sequential API calls. The scheme of the generation of the dataset can be seen in Figure 1.5. As this dataset is the newest and contains Windows-related API calls sequences, it is similar to the ADFA-IDS family datasets. Main disadvantage is that collected system calls have no information about passed parameters and return values. Other researchers tend to generate private datasets. For example, Kolosnjaji et al constructed deep neural networks to improve modelling and classification of system call sequences. By combining convolutional and recurrent layers in one neural network architecture, they obtained optimal classification

results (average accuracy of 89.4%) (Kolosnjaji et al., 2016). But again, this dataset contained only system calls without any supporting data and was used only for that particular research.

Besides API sequences, static PE information-containing datasets are popular among malware researchers. Most researchers are generating private datasets intended for one-time usage. However, one quite popular public dataset that contains static malware data exists. In 2015 Microsoft created a Kaggle competition named “Microsoft Malware Classification Challenge”. For that particular event, a special dataset has been created (Big 2015). The raw data of dataset contains the hexadecimal representation of the file's binary content. Additionally, Microsoft provided a metadata manifest, which is a log containing various metadata information extracted from the binary, such as function calls (not a system calls sequences, only functions used by the binary file), strings, etc. The dataset contains nearly 0.5 terabytes of data, of more than 20K malware samples. Apart from serving in the Kaggle competition, this dataset has become a standard benchmark for research on modelling malware behaviour (Ronen & Feuerstein, 2015).

Table 1.9. Comparative information of reviewed datasets suitable for intrusion and malware detection

Dataset	Year	Operating system	Notes
DARPA 98	1998	Linux	Contains extensive network information when several attacks have been executed. Collected data is old and does not meet today's standards.
KDD Cup 99	1999	Linux and Windows	Contains extensive network and Windows NT audit log data. Collected data is old and does not meet today's standards.
FirefoxDS	2013	Linux	System calls of Firefox application under attack are collected.
ADFA-LD	2014	Linux	Extensive host-based information (system calls)
Berlin et al., 2015	2015	Windows	Windows audit log from malware has been collected. No System call information. System calls not mentioned directly in the research and collected dataset is not public.
Big 2015	2015	Windows	Collected static Windows PE executables information.

End of the Table 1.9

Dataset	Year	Operating system	Notes
Kolosnjaji et al., 2016	2016	Windows	Collected system calls of malware samples. Dataset is private and contained only system calls without additional data (e.g., parameters and return values)
ADFA-WD	2016	Windows	Collected host-based information – system calls. Basic information (only system calls names) was collected, and an insufficient amount of vulnerabilities was used to generate malicious activity.
NGIDS-DS	2017	Linux	Synthetically created dataset. Contains HIDS and NIDS related data. HIDS data contains only Linux machine information.
CSE-CIC-IDS2018	2018	Linux and Windows	Main scope of dataset – network related data. Host-based data only presented, but not used in any experiments.
Mal-API-2019	2019	Windows	Dataset contains malware generated API calls. Collected system calls have no information about passed parameters and return values.

To close the gap for the Windows HIDS and AV compatible dataset, we have proposed the AWSCTD dataset, that contains labelled system calls sequences in the form of univariate times series, generated on Windows host machine in the presence of malware and clean applications samples, as well as corresponding network traffic, modifications made to the registry data and file system (D. Čeponis & Goranin, 2018). Still, the main strength of AWSCTD dataset is a huge collection of system calls that, as it was stated earlier, can be considered as the most reliable source for intrusion and malware detection, since they represent low level non-filtered and unprocessed information. AWSCTD, together with ADFA-IDS, CIC-IDS2018 and Mal-API-2019 can be considered as the most comprehensive and based on modern host-level cyber-attacks. In comparison, Big 2015, Berlin et al. and Mal-API-2019 datasets contain much higher number of executed malicious/benign applications than novel AWSCTD dataset. But they have several limitations: Berlin et al. dataset is privately collected and not published, and they either only contain static malware data or system calls without parameters information. It is also necessary to consider that MAL-API-2019 has

appeared two years later than AWSCTD. Comparative information of datasets suitable for intrusion and malware detection can be seen in Table 1.9.

1.3.4. Application of Artificial Neural Networks and Deep Learning in Intrusion and Malware Detection Systems

The first advantage in the use of a neural network in intrusion detection would be the flexibility that the network would provide. A neural network would be capable of analysing the data from the network, even if the data is incomplete or unclear. Similarly, the network would possess the ability to conduct an analysis with data in a non-linear fashion. Further, because some attacks may be conducted against the network in a coordinated attack by multiple attackers, the ability to process data from a number of sources in a non-linear fashion is especially important. The natural speed of neural networks in classification (learning stage is still a disadvantage) is another advantage (Iftikhar Ahmad et al., 2008).

One of the first works in intrusion detection by NN was performed in 1998 (Ryan et al., 1998). Authors trained and tested an offline NNIDS on a system of ten users. They used 2-Layer MLP architecture for their system and backpropagation for training purpose. The data source for training and testing was operating system logs in a UNIX environment. The result parameters to evaluate the performance of the system were false positive and false negative.

Another attempt in the same field also was made in 1998 (Cannady, 1998). He also used the 2-Layer MLP architecture for his system and backpropagation for training purpose. The data source for training and testing was Network Packets collected by Real Secure. Nine of the packet characteristics of network data were selected and presented to the MLP network, which has four fully connection layers. He used RMSE (root mean square error) parameter for training and testing data for performance measuring.

In 1999 a host-based IDS that focused on building program profiles and used these program profiles to identify normal software behaviour and malicious software behaviour was presented (Schwartzbard, 1999). The system was trained and tested on SUN platform and used Basic Security Module (BSM) as a source of data. Input data were extracted from BSM and a distance metric, which constituted input vectors of the NN. The IDS presented was a single hidden layer MLP. The number of input nodes was equal to the number of exemplar strings. Lucky Bucket algorithm is used to capture the temporal locality of anomalous events. Performance analysis was done with KDD cup dataset. Same authors in 1999 also used Elman Networks (recurrent neural networks) (Ghosh et al., 1999).

Another study was done in 2000. It proposed to use of self-organizing neural networks (SONN) to recognize anomalies in the network data stream (Rhodes et al., 2000). Unlike approaches using self-organizing maps to process the entire

state of a network or computer system to detect anomalies, the proposed system breaks down the system by using a collection of more specialized maps. A monitor stack was constructed, making each neural network a kind of specialist in recognizing the normal behaviour of a protocol and raising the alarm when a deviation from normal profile occurs. The test intrusions were buffer overflow attack attempt.

Next, in 2000, a misuse detection model was introduced for searching attack specific keywords in the network traffic (Richard P. Lippmann & Cunningham, 2000). It was used a MLP network to detect Unix-host attacks, and attacks to obtain root-privilege on a server. The data that they presented to the neural network consisted of attack-specific keyword counts in network traffic. Two neural networks were used in the system, one for providing an attack probability and another one for classifying attacks. A two-layer perceptron was designed with k input nodes, $2k$ hidden nodes and 2 outputs (normal and attack) and backpropagation was used as a training algorithm.

In 2001, a statistical analysis was used in conjunction with MLP networks (Z. Zhang et al., 2001). The probe collects network traffic and abstracts it into statistical variables. Event pre-processor collects data from probes and other agents and formats it for the statistical analyser. The statistical model compares the data to the previously compiled reference model, which describes the normal state of the system. A “stimulus vector” is formed and forwarded to the NN. Neural network analyses the vector and decides whether it is anomalous or normal. The experimental test bed consisting of 11 workstations and one server was built by using OPNET network simulation software. UDP flooding attack was simulated within the test bed.

Later in 2001 an experimental IDS with a hierarchy of neural networks was introduced. Each of the neural networks in the hierarchy focused on different portions of nominal TCP behaviour. Portions of these observed TCP behaviours are connection establishment, connection termination and port usage. The system was trained to detect three types of attack: SYN flood, fast SYN port scan, and stealth SYN port scan. Backpropagation learning algorithm was used to train system. Input vectors to each of the neural networks were generated randomly (S. C. Lee & Heinbuch, 2001).

In 2002, an alternative methodology for both visualizing intrusions by using SOM and classifying intrusions was presented (Jirapummin et al., 2002). It used Resilient Propagation, Neptune attack (SYN flooding), Portsweep and Satan attacks (port scanning) from KDD Cup 1999 data set. For RPROP, 3-layer NN is utilized with 70 nodes in the first hidden layer, 12 neurons in the second hidden layer and 4 neurons in the output layer. The transfer functions for the first hidden layer, second hidden layer and the output layer of RPROP were tan-sigmoidal, log-sigmoidal and log-sigmoidal, respectively.

In 2002 a neural network model for a network-based intrusion detection system was proposed (Bivens et al., 2002). Anomaly detection system had used MLP network for detection. System read tcpdump data from a KDD cup 99 dataset.

Later, in 2003, a novel scheme that uses robust principal component classifier in intrusion detection problems where the training data may be unsupervised was proposed (Shyu et al., 2003). Assuming that anomalies can be treated as outliers, an intrusion predictive model is constructed from the major and minor principal components of the normal instances. A measure of the difference of an anomaly from the normal instance is the distance in the principal component space. The distance based on the major components that account for 50% of the total variation and the minor components whose eigenvalues less than 0.20 is shown to work well. The experiments with KDD Cup 1999 data demonstrate that the proposed method achieves 98.94% in recall and 97.89% in precision with the false alarm rate 0.92% and outperforms the nearest neighbour method, density-based local outliers (LOF) approach, and the outlier detection algorithm based on Canberra metric.

Next, in 2004 Unsupervised Neural Net based Intrusion Detector (UNNID) system was proposed, which detects network-based intrusions and attacks using unsupervised neural networks (Amini & Jalili, 2004). The system had facilities for training, testing, and tuning of unsupervised nets to be used in intrusion detection. Using the system, we tested two types of unsupervised Adaptive Resonance Theory (ART) nets (ART-1 and ART-2). Based on the results, such nets can efficiently classify network traffic into normal and intrusive. The system uses a hybrid of misuse and anomaly detection approaches, and is therefore capable of detecting known attack types as well as new attack types as anomalies. KDD Cup was used for evaluation.

In 2005 an MLP(Multi-Layer Perceptron)/Elman neural network was proposed, which realizes classification with memory of past events using the real-time classification of MLP and the memorial functionality of Elman (Yu et al., 2005). The system's sensitivity to the memory of past events can be easily reconfigured without retraining the whole network. This approach can be used for both misuse and anomaly detection system. The intrusion detection systems using the hybrid MLP/Elman neural network was evaluated using KDD Cup dataset.

In 2007, a resilient backpropagation usage in intrusion detection system was proposed (I Ahmad et al., 2007). The system also used famous KDD Cup dataset. The same authors in 2008 wrote a paper about different backpropagation algorithms benchmarking (Iftikhar Ahmad et al., 2008). Dataset – KDD Cup.

Hierarchical intrusion detection model using Principal Components Analysis (PCA) method was introduced in 2007 (G. Liu et al., 2007). Most of the existing intrusion detection (ID) models with a single-level structure can only detect either

misuse or anomaly attacks. In the proposed model, PCA is applied for classification, and neural networks are used for online computing. Experimental results and comparative studies were based on the 1998 DARPA provided KDD Cup dataset.

Wavelet Neural Network-based intrusion detection method was introduced in 2009 (Sun et al., 2009). It reduces the number of basic wavelet functions by analysing the sparseness property of the sample data to optimize the wavelet network. The learning algorithm trains the network using gradient descent. DARPA dataset was used for testing purposes.

The recurrent neural network was combined with an expert system in 2010 to classify network data into anomalous and normal (Yong & Feng, 2010). Authors developed an incremental kernel principal components algorithm to pre-process the data that goes into the neural network. KDD Cup 99 dataset was used in the testing phase.

Fuzzy clustering combination with neural networks was proposed in 2010. The solution was called FC-ANN. It was based on ANN and fuzzy clustering, to solve the problem and help IDS achieve higher detection rate, reduce false positive rate and improve stability (G. Wang et al., 2010). The general procedure of FC-ANN is as follows: firstly, the fuzzy clustering technique is used to generate different training subsets. Subsequently, based on different training subsets, different ANN models are trained to formulate different base models. Finally, a meta-learner, fuzzy aggregation module, is employed to aggregate these results. Experimental results on the KDD CUP 1999 dataset show that proposed new approach, FC-ANN, outperforms BPNN and other well-known methods such as decision tree, the naïve Bayes in terms of detection precision and detection stability.

In 2013, the Discriminative Restricted Boltzmann Machine was applied to network anomaly detection in a semi-supervised fashion (Fiore et al., 2013). This one solution belongs to the deep learning class. KDD Cup data also was used for testing and evaluation. It is worth to mention that training times were significant for this proposal. Training in Experiment 1 over 15 epochs took 186.4 h on a HP DL380 G3 server. This type of server was introduced in 2005.

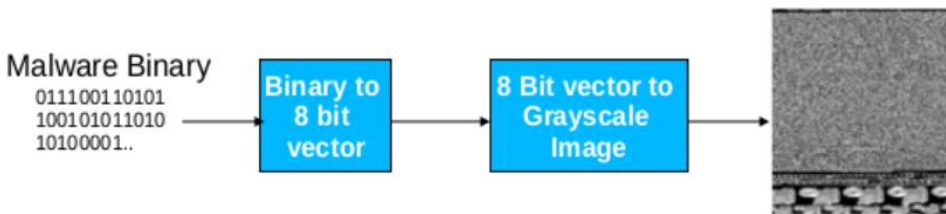


Fig. 1.6. Visualizing Malware as a Gray-Scale Image (Gibert, 2016)

DL methods for intrusion detection began being used in 2014 when a deep belief network (DBN) has been used as a classifier with the combination of restricted Boltzmann machine (RBM) for training and backpropagation for the fine-tuning process of KDDCup 99 (Gao et al., 2014). Later, when computer processing (especially GPU) capabilities were improved – more complex DL configurations have emerged. RNN networks were the most apparent methods to adapt for intrusion detection due to the ability to work with time-series data (J. Kim & Kim, 2016; Maulana & Kusuma, 2020; Staudemeyer, 2015; Tang et al., 2018).

CNN networks for malware classification started to be used in 2016 (Gibert, 2016; Kolosnjaji et al., 2016). Gibert used Microsoft generated BIG 2015 dataset and proposed to create images from malware binary files. An example of image generation can be seen in Figure 1.6. Later, the image created has been used with various CNN models in the classification tasks. The accuracy of 0.9976 has been achieved with model CNN 2C 1D (Figure 1.7).

Kolosnjaji et al used a privately generated dataset of malware initiated system calls. System calls sequences has been used by DL network consisting of CNN and LSTM layers (Figure 1.8). Proposed model outperformed the feedforward network in terms of accuracy (79.8% versus 89.4%). Proposed model outperformed not only other simpler neural architectures, but also previously widely-used Hidden Markov Models and Support Vector Machines.

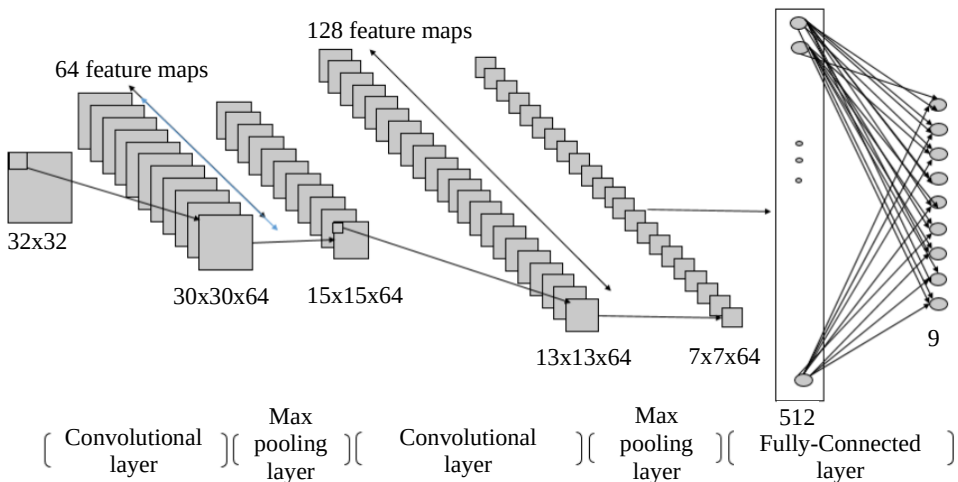


Fig. 1.7. Malware classification model CNN 2C 1D (Gibert, 2016)

The first concept of CNN usage for the intrusion detection has been proposed in (Paper, 2016) and was later successfully applied by (Dawoud et al., 2018;

De Teyou & Ziazet, 2019) with the results up to 99% of network anomaly detection accuracy.

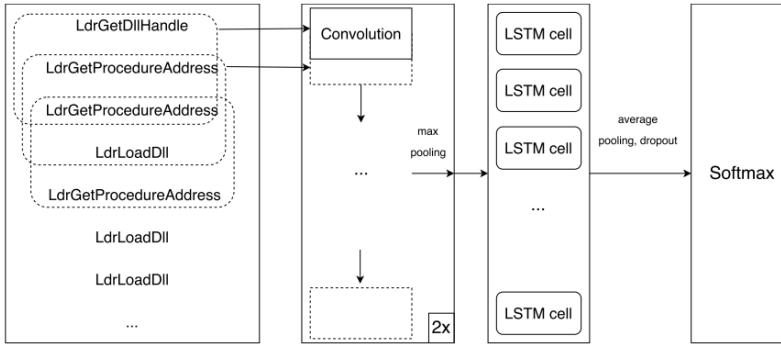


Fig. 1.8. Deep Neural Network architecture for system calls sequences classification (Kolosnjaji et al., 2016)

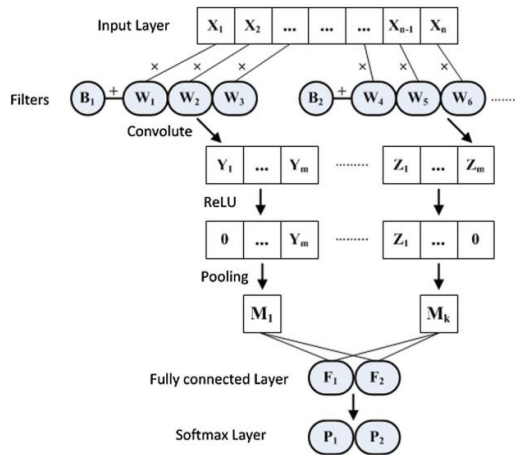


Fig. 1.9. CNN model architecture for HIDS (Tran et al., 2018)

Afterwards, DL methods have been widely used for the datasets mentioned above. CNN for HIDS has been tested with various sliding window sizes on ADFA-IDS and NGIDS-DS (Tran et al., 2018). The results were somehow better than the counterpart of the work in (Xie, Hu, Yu, et al., 2014), where classical machine learning methods were applied, with a detection rate (DR) of 60% and False Alarm Rate (FAR or also known as FPR) of 20%. To achieve these results

the input layer of 191 and 151 system calls was used. How can it be seen in Figure 1.9 – relatively simple CNN configuration has been used in experiments (only one layer of convolutions has been applied). The short sequence of system calls, and simple CNN model resulted in not so great results: the detection rate of 60.46% for NGIDS-DS dataset.

Other authors have used dCNN not to perform final classification but to extract useful features from the data, and the Softmax classifier to generate the final detection result for the ADFA-IDS dataset. This method has enhanced the real-time processing and detection efficiency of intrusion detection systems by 7.3 percent (S. Zhang et al., 2019).

In recent years, the research of the combination of CNN and RNN into one model has shown promising results (Guo et al., 2018; Ji et al., 2019; J. Wang et al., 2016) and was applied from image recognition to Bitcoin price prediction tasks. This encouraged many researchers to apply similar methods in the IDS research field. The combination of CNN and RNN was applied in NIDS and HIDS: it was tested with KDDCup 99 (Vinayakumar et al., 2017) and ADFA-LD (A. Chawla et al., 2018) datasets, and comparable results have been achieved. The main disadvantage of such an approach is that proposed models are complicated and require massive computational power. All comparative information on the aforementioned research can be seen in Table 1.10.

Table 1.10. Comparative information on reviewed ML and DL methods usage in intrusion and malware detection

Reference	Year	Method used	Notes
Ryan et al., 1998	1998	NN	Used 2-Layer MLP. Used UNIX system log data. A legitimate user was identified 92% of the time.
Cannady, 1998	1998	NN	Used 2-Layer MLP. Used network-related data. RMSE parameter used to evaluate NN. The training returned 0.058298 value of RMSE.
Schwartzbard, 1999	1999	NN	Host data was used (software behaviour profiles). A single hidden layer MLP was used. 90.9% of all intrusions were detected correctly, with a false positive rate of 18.7%.
Rhodes et al., 2000	2000	SONN	Network data were used to catch an intrusion.
Richard P. Lippmann & Cunningham, 2000	2000	NN	MLP network was used on network-level Unix data (DARPA 1998 dataset). Designed a system to detect attacks with a possibility 80% of attacks with only one false positive per day.

Continue of the Table 1.10

Reference	Year	Method used	Notes
Z. Zhang et al., 2001	2001	NN	Statistical analysis was used in conjunction with MLP network with network-related data. Proposed framework to analyse network data with the help of NN.
S. C. Lee & Heinbuch, 2001	2001	NN	Hierarchy of NN's were used in experimental IDS. Network data was used in an experiment. The experimental IDS was 100% successful in detecting specific attacks, without a priori information on or training directed toward those attacks.
Jirapummin et al., 2002	2002	NN	Network data from KDD Cup 1999 dataset was used. A 3-layer NN was used. Researchers achieved more than 90 % detection rate and less than 5 % false alarm rate in three selected attack programs.
Bivens et al., 2002	2002	NN	Proposed a NN model for a NIDS. Used KDD Cup 1999 dataset. The NN performed well when tested on individual types of attacks one at a time. But failed to perform simultaneous attacks detection.
Shyu et al., 2003	2003	NN	Used KDD Cup 99 dataset. Achieved 98.94% recall in intrusion detection task.
Amini & Jalili, 2004	2004	NN	Unsupervised NNs were used to detect network-based intrusion. Used KDD Cup 99 dataset. The results shown that proposed method can recognize attack traffic from normal one in 90.7–93.5 percent of times.
Yu et al., 2005	2005	NN	An MLP/Elman model was used. Used KDD Cup 99 dataset. The detection rate of 87.9% achieved with a false positive rate of 22.4%.
I Ahmad et al., 2007	2007	NN	Used NN with a Resilient Backpropagation. Used KDD Cup 99 dataset. The overall detection rate was 95.93 %.
G. Liu et al., 2007	2007	NN	NNs were used for online computing. Used DARPA 1998 dataset. The anomaly detection method proposed a 97.1% detection rate with 2.8% FPR.
Sun et al., 2009	2009	NN	A wavelet NN was used. Used DARPA 1998 dataset. The training and testing error values were smaller than regular NN.

Continue of the Table 1.10

Reference	Year	Method used	Notes
Yong & Feng, 2010	2010	NN	A recurrent MLP network was used. Used KDD Cup 99 dataset. An average detection rate of 82% was produced for network related attacks.
G. Wang et al., 2010	2010	NN	A NN combined with fuzzy clustering was used. Used KDD Cup 99 dataset. An average accuracy of 96.71% was produced with the proposed NN configuration.
Fiore et al., 2013	2013	NN	A restricted Boltzmann machine network was used. Used KDD Cup 99 dataset. An accuracy of 84% produced when KDD dataset was used for training.
Gao et al., 2014	2014	DBN	A DBN used in combination with RBM. Used KDD Cup 99 dataset. An accuracy of 93.49% and FPR of 0.76% was produced by the proposed network.
Gibert, 2016	2016	CNN	A static data from BIG 2015 dataset was used with CNN. An accuracy of 99.76% was achieved by the proposed model.
Kolosnjaji et al., 2016	2016	CNN + RNN	A combined model of CNN and LSTM was used with a privately generated malware-based dataset. An accuracy of 89.4% was achieved by the proposed model.
Paper, 2016	2016	CNN	A CNN model was used for training. Used KDD Cup 99 dataset. An error lower of 2% for training and testing was produced.
Vinayakumar et al., 2017	2017	CNN + RNN	A CNN, LSTM and GRU combinations used. Used KDD Cup 99 dataset. The best accuracy of 98.7% was proposed by a complicated CNN 3 layer-LSTM model.
Tran et al., 2018	2018	CNN	A relatively simple CNN architecture was used with ADFA-IDS and NGIDS-DS datasets. An accuracy of 60% and FPR of 20% was generated.
A. Chawla et al., 2018	2018	CNN + RNN	A combined CNN and GRU model was used with ADFA-LD dataset. A high detection rate of 100% with PFR of 60% was produced by the proposed model.

End of the Table 1.10

Reference	Year	Method used	Notes
S. Zhang et al., 2019	2019	RNN	A 3-layer LSTM-based model was used with network-level USTC-TFC2016 dataset. An accuracy of 99.9% was achieved when the network-level intrusion was simulated.

1.4. Application of System Calls for Intrusion and Malware Detection

The intrusion threats or obfuscation techniques used by malware can be combated by collecting system calls and their further analysis. The behaviour of a program and its security-relevant activity is tightly related to system calls used that can be analysed to find clusters of samples that behave similarly. Because of these reasons, system calls should be collected and stored in a secure environment for analysis. System call tracing is a popular source of data not only for dynamic malware detection and classification tasks but also for early execution prevention of unwanted (malicious) applications (Pailoor et al., 2020). The IDS and malicious applications detection are deeply related because the use of system call tracing for malware detection has originated from intrusion detection, where unusual system call sequences were triggered as evidence of potential irruptions (Stephanie Forrest et al., 1998). Therefore, a system call is the best way to trace software activities on the operating system level. The dynamic analysis is used to determine the behaviour of programs while executing them in a safe and isolated environment. When the program is executed, it throws system call invocations. The detection method is based on monitoring the system calls in a fast way to understand the objectives of the suspicious program (Canzanese et al., 2016). An inability to use system calls as the data source is considered as one of the main disadvantages of the modern IDS. Furthermore, the usage of system calls can reduce FPR and response time of the systems (Singaravelan et al., 2020; Wunderlich et al., 2020).

Two methods are typically used to analyse system calls: frequency-based and sequence-based. The first proposition to use system calls as information for intrusion was based on the statement that normal (frequent) sequences differ from abnormal (infrequent) ones (S. Forrest et al., 1996). System calls are treated as “words” and sequences of them – as “sentences”. This type of intrusion and malware detection requires relatively high computational resources, but in favour, it returns the highest accuracy results. Hu et al. proposed a simple and efficient HMM scheme for host-based anomaly intrusion detection (Hu et al., 2009). They proposed an enhanced incremental HMM training framework that incorporates a

simple data pre-processing method for identifying and removing similar sub-sequences of system calls. Ye et al. proposed the usage of rough set theory in the area of intrusion detection to make it more suitable as a detection system (Ye et al., 2010). The study confirmed that it is possible to detect an attack by system call sequence analysis. Jewell & Beaver shown that system call sequences can be found to reach a steady state across processes and users, and explore the viability of new methods as heuristics for profiling user behaviours (Jewell & Beaver, 2011). The paper of Creech & Hu has clearly demonstrated the superior results possible when using a full semantic analysis of system calls to derive a new feature. The extreme learning machine methodology has been verified as applicable to the IDS problem (Creech & Hu, 2014). The paper of Gupta & Kumar describes an approach to detect malicious program executions in Cloud environment with the use of a new technique of Immediate System Call signature detection. In this approach, for all unique System Calls, the Immediate System Call structure is identified and created from their normal execution logs, and such signatures are stored and then used as a baseline for anomalous program detections (Gupta & Kumar, 2015). The aforementioned research of Kolosnjaji et al. proposed to use system calls sequences as a feature vector for the malware classification (Kolosnjaji et al., 2016). They get concentrated on the DL model construction and achieved better classification results than not only other simpler neural architectures but also previously widely used Hidden Markov Models and Support Vector Machines. One of the latest paper proposes to use natural language processing to detect malicious behaviour in Windows virtual machines by analysing system calls sequences (Peddoju et al., 2020).

Frequency-based intrusion and malware detection methods are often used for the computationally inexpensive characteristic. Several frequency-based methods have been applied to the already mentioned ADFA-LD dataset. It has been considered simple features such as a trace's length and the relative frequency of each call in that trace, and achieved "acceptable" detection results in testing using simple one-class classifiers: k-NN, SVM, and k-means algorithms (Xie, Hu, & Slay, 2014; Xie, Hu, Yu, et al., 2014; Xie & Hu, 2013).

System calls parameters, along with memory pointers, also are proposed as an additional data source to detect intrusion or malware. For example, Abad et al. described an IDS based on correlating network traffic to system calls (Abad et al., 2003). They were able to make abnormal traces more abnormal, thus making intrusions easier to detect. Kruegel et al. introduced several models that learn the characteristics of legitimate parameter values and are capable of finding malicious instances (Kruegel et al., 2003). Based on the proposed detection techniques, they developed a host-based intrusion detection tool that monitors running applications to identify malicious behaviour. Tandon Gaurav & Philip K. Chan portrayed the

efficacy of incorporating system call argument information and used a rule-learning algorithm to model a host-based anomaly detection system (Tandon Gaurav & Philip K. Chan, 2005). Based on experiments on various data sets, they claim that the proposed argument-based model detected more attacks than all tested sequence-based techniques.

Contrary exists techniques that can be used against such detection methods when system calls are used. Typically, these two key ideas are used: “mimicry” attacks and the usage of short malicious sequences. In the first case, system calls with nulls as parameters are used to imitate benign application (Kayacik & Zincir-Heywood, 2008). Likewise, malicious actions are executed not in one atomic operation but in short sequenced bursts hidden in a valid system calls sequences (Tan & Maxion, 2002). One of the most dangerous malware type ransomware was analysed by Ahmed et al. They proposed a non-signature-based method for the detection of the ransomware on windows API call sequences. This paper introduced a refinement process that reduces the size of the system call traces collected from the dynamic analysis logs. This approach filtered out the program's traces by removing the system calls that do not have a strong indication of the behaviour of the ransomware (Ahmed et al., 2020). Summary of reviewed application of system calls for intrusion and malware detection can be seen in Table 1.11.

Table 1.11. Summary of reviewed application of system calls for intrusion and malware detection

Reference	Year	Method used	Notes
S. Forrest et al., 1996	1996	Sequence-based	The first proposition to use system calls as data. Used short sequences of system calls of Linux applications to detect intrusion.
Abad et al., 2003	2003	System calls parameters	The authors proposed a proof of concept method for intrusion detection by making abnormal system calls traces more abnormal, thus making intrusions easier to detect.
Kruegel et al., 2003	2003	System calls parameters	Several, working with system calls parameters, models proposed for intrusion detection. An HMM model used with Linux-based data.
Tandon Gaurav & Philip K. Chan, 2005	2005	System calls parameters	An argument-based model, A-LERAD introduced and tested with various datasets (Linux-based). The proposed methods detected more attacks than all tested sequence-based techniques.

End of the Table 1.11

Reference	Year	Method used	Notes
Hu et al., 2009	2009	Sequence-based	Proposed a method for identifying and removing similar sub-sequences of system calls.
Ye et al., 2010	2010	Sequence-based	Proposed a method on how to reduce the dimensionality of the data when system calls are used for intrusion detection.
Jewell & Beaver, 2011	2011	Sequence-based	System calls analysis applicability was tested against data theft from the Linux operating system.
Xie & Hu, 2013	2013	Frequency-based	A preliminary analysis of ADFA Linux dataset proposed to use frequency-based intrusion detection. The authors proposed to use (kNN)-based model.
Creech & Hu, 2014	2014	Sequence-based	Demonstrated system calls sequence-based methods for intrusion detection. A Linux-based ADFW dataset has been used.
Xie, Hu, & Slay, 2014	2014	Frequency-based	A novel SVM-based anomaly detection system is proposed for ADFA-LD.
Xie, Hu, Yu, et al., 2014	2014	Frequency-based	Analysed several frequency-based algorithms in attempts to reduce the dimension of the frequency vectors. Linux based ADFA dataset have been used.
Gupta & Kumar, 2015	2015	Sequence-based	Demonstrated system calls sequences usage for cloud-based intrusion detection.
Kolosnjaji et al., 2016	2016	Sequence-based	The proposed system calls sequences for malware classification. Worth to mention, that DL model was used for such a task.
Peddoju et al., 2020	2020	Frequency-based	Proposed two anomaly detection algorithms for intrusion detection with system calls.

1.5. Intrusion and Malware Detection Research Situation in Lithuania

As intrusion and malware research is popular in the global academic world, Lithuanian researchers do not generate a noticeable amount of papers in this area. Especially in host-based area. All recent year's research has been dedicated to network-based research with a combination of popular ML methods.

Paulauskas & Bagdonas presented a novel approach to detect network flow anomalies. The method is based on the Local Outlier Factor (LOF) algorithm, which evaluates each event's uniqueness on the basis of distance from the k -nearest neighbours (Paulauskas & Bagdonas, 2015). Later, research has been continued by the application of the LOF algorithm for the detection of anomalies when only normal network data are used for the model training has been demonstrated. Experimental results of different threshold values influence on the anomaly detection accuracy using NSL-KDD dataset is presented (Auskalnis et al., 2018). The most recent LOF usage in intrusion detection was presented by Paulauskas & Baskys: they analysed the use of the histogram-based outlier score (HBOS) to detect anomalies in the computer network. Experiments were conducted using an NSL-KDD dataset (Paulauskas & Baskys, 2019).

Bulavas, in 2018, analysed data visualization methods usage for intrusion detection. Investigation shows that a combination of Principal Component Analysis (PCA) and Decision Tree methods allow classification of intrusions such as Smurf, Satan, Neptune, Portsweep, Ipsweep with probabilities higher than 95% with a depth of tree set to 4 and PCA components set to 10. Nevertheless, Nmap, Teardrop intrusions are classified purely, therefore, a deeper Decision Tree is needed to increase classification accuracy (Bulavas, 2018).

In the same 2018 year, Japertas & Baksys proposed a method for early staged detection of pre-planned attacks using a number of logical filters. The results of theoretical simulation have shown that the proposed method is capable of determining early-staged cyberattacks (Japertas & Baksys, 2018).

In 2019, Maskeliūnas and Damaševičius presented a method to detect intrusion in the network level. This research proposes a solution for intrusion detection by using the improved hashing based Apriori algorithm implemented on Hadoop MapReduce framework. The obtained results have shown that this approach provides a reliable and effective means of detecting network intrusion (Azeez et al., 2019). The detection rate has been achieved up to 98.2% when a DOS attack was classified. The same authors participated in research where phishing attacks detection were analyzed, and a new method to prevent internet users from such vulnerability has been proposed (Azeez et al., 2020). Damaševičius and Blažauskas cooperated in research where malware classification task has been analyzed. They proposed a feature fusion method to combine the features extracted from pre-trained AlexNet and Inception-v3 deep neural networks with features attained using segmentation-based fractal texture analysis (SFTA) of images representing the malware code (Nisa et al., 2020). The presented method was evaluated on a Maling malware image dataset, achieving an accuracy of 99.3%, which makes it the best among the competing approaches.

It is worth to highlight the work of Venčkauskas et al., where BitTorrent Sync client encryption analysis has been used to help the forensics team in a decision

where the shared data is legal or not. They proposed the methodology that enables to capture the initiation step and to inform the forensics investigator, which then takes the reactive actions. The comparison of the calculated hash with the list of publicly available hashes allows determination of whether sharing of the file is legal or illegal (Venckauskas et al., 2017).

1.6. Conclusions of Chapter 1 and Formulation of the Objectives of the Thesis

The following conclusions can be made from current chapter information:

1. The protection of information systems is one of the most important tasks for modern companies. This is proven by both industrial and scientific publications. If this task is failed to perform correctly, the company can lose important data or even critical infrastructure. IDSs are used as one of the protective layers for such task: NIDS and HIDS are the most popular approaches with no winner for all types of protection;
2. Classical intrusion and malware detection approach based on signatures has a number of flaws. The significant number of false-negatives can be considered as the main weakness of this approach and motivates further research on ML/DL application for anomaly-based detection, that could potentially deal with zero-day and evolving attacks. On the other hand, application of such methods is highly dependent on the quality and amount of data that is used for method training. ML/DL methods for IDS usually achieve high accuracy (66.46–99%) only when complicated models are used in combination with high dimensionality data;
3. It was determined by the analysis performed, that sequences of system calls invoked by malware and exploits used for the attack on the system level are seen as one of the most perspective sources for intrusion and malware detection, since sequences of calls by malicious and benign programs differ in nature. Malware in this thesis is referenced to a virus which is described as “a program that is able to infect other programs by modifying them to include a possibly evolved copy of itself” (Szor, 2005). Furthermore, this thesis does not analyse network-level attacks (e.g., DDOS) but concentrates on attacks (viruses) or attack tools that utilise integrated OS level API's;
4. The wider existence of network-based datasets is steering researchers to develop more methods for NIDS. This situation has also encouraged researchers to create several HIDS-based datasets containing system calls

from Windows OS. Unfortunately, the analysis revealed that existing datasets have several flaws: they lack essential features (ADFA-WA contains only system calls identifiers), have an insufficient amount of traces (ADFA-WA has only 6636 traces), or are only presented, but not used in any experiments (CSE-CIC-IDS2018 dataset). The analysis performed has shown that there is an increasing requirement for the development and training of anomaly-based HIDS solutions, which is currently being slowed down by the lack of available and suitable host-level anomaly datasets;

5. It can be concluded that the research on improvement of anomaly-based host-level intrusion and malware detection by applying ML and DL methods should be concentrated on the creation of an exhaustive dataset of system calls sequences by malicious and benign applications with further application of the created dataset for selecting and training the most appropriate ML/DL models.

Based on the conclusions, the following tasks are formulated to achieve the goal:

1. To propose robust method for generating a dataset that can be used as a basis for further research of ML and DL methods application for host-level intrusion and malware detection task;
2. To perform an experimental evaluation of existing ML and DL methods with newly generated dataset for the host-level intrusion and malware detection task;
3. To propose and evaluate new DL models for the host-level intrusion and malware detection task.

Generation of Host-Based Windows Dataset for Anomaly-Based Intrusion Detection

This Chapter provides a description of the dataset generation method, practical method implementation description, including the process of malware preparation for the experimental environment. The detailed specification of the newly generated dataset Attack-Caused Windows OS System Calls Traces Dataset (AWS-CTD), including size, structure, characteristics, balancing methods applied are presented. Apart from that, the evaluation criteria of ML/DL methods are defined that will be later used for methods efficiency evaluation.

The proposed method and results were published in the international journal and presented in an international conference (Čeponis & Goranin, 2018).

2.1. Dataset Generation Method Description

The lack of available and suitable host-level anomaly datasets has caused the need to create such a dataset from scratch. The system engineering approach was applied in order to cover the research needs while developing the dataset, including requirements formulation, UML diagrams and other methods.

The following non-functional requirements were formulated for the malicious activity dataset generation method: the system has to be flexible (it must allow adapting new data collection in the future), easy to configure (no special tools must be required to change system parameters), and based on open-source software only. The target operating system for malicious activity collection chosen – was Windows, because it is still the most widely used operating system in the world, although the method can be easily adapted for any other OS.

For reasons of simplicity and proof of concept, only openly available malware samples were used to generate malicious activity samples. The method can be easily automated: any malware samples can be downloaded, prepared according to the requirements, and used. Malware samples contain items for all well-known operating systems, so malicious activity sets are generated for any operating system.

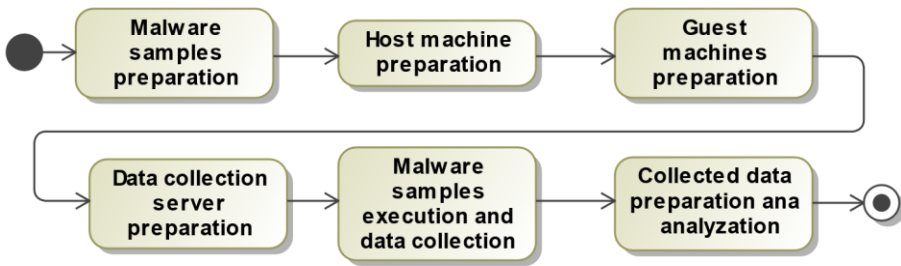


Fig. 2.1. Main steps of the proposed method of dataset generation

The proposed method (Figure 2.1) has six following steps:

1. **Malware samples preparation.** At first, malware must be obtained from available sources. Later, malware of Windows OS executables type should be extracted and added to a separate collection for use;
2. **Host machine preparation.** The hypervisor must be installed and configured on the selected host machine. Malware samples must be copied to the host machine for later execution;
3. **Guest machines preparation.** Template for a virtual guest machine must be added and configured on the host machine. Later, the required number of guest machines (copies) should be created for malware execution. Execution of malware samples can be performed in parallel and is dependent on the number of guest machines available;

4. Data collection server preparation. Server storage for the malware execution log information (such as anomaly samples in the form of logs, network activity, system calls, etc.) must be prepared;
5. Malware samples execution and data collection. When all machines are prepared – the main execution script is started. The main script will upload a malware sample to the target guest machine and will start the operating system. Later, a script on the guest machine will execute the malware on OS start-up. The malware-generated activity log will be automatically collected and uploaded to the data collection server;
6. Collected data preparation and analyzation. When all samples are executed, collected data can be transformed into the XML format and analysed.

2.2. Dataset Generation Method Implementation

The proposed method implementation (architecture) can be seen in Figure 2.2. The virtualization technique, based on a free ProxMox hypervisor, was selected to simulate quest machines that will be used for running malicious actions. ProxMox VE is a completely open-source platform for enterprise virtualization, a built-in web interface that allows management of VMs and containers, software-defined storage and networking, high-availability clustering, and multiple out-of-the-box tools in a single solution (Kovari & Dukan, 2012). ProxMox is running QEMU - a generic and open source machine emulator and virtualizer and is based on Debian operating system. According to the results of the latest research, QEMU has less detectable virtualization through basic detection techniques (Miller et al., 2017), which maximizes the malware execution rate.

The main bash script is executing all commands required to collect data: a guest machine is prepared, started and stopped by that script. The main bash script has only one parameter – a folder that contains prepared malware samples. ProxMox firewall is enabled on the Host machine to manage network flow and minimize the risk of malware propagation. Only one-directional flow to the remote HIDS server was allowed – all other connections were blocked. All data sent to that server was stored on LOG server for later analysis.

An anomaly data collection was done by three tools: Dr. Memory provided system call tracer for the Windows OS, OSSEC (open source HIDS (Timofte, 2008)) for file integrity monitoring and WinDump for the network-related information. Dr. Memory tool provides not only the system call name but also passed parameters list and return value. All that information can be used to detect earlier

mentioned thread injection, which is missing in the method provided by (Berlin et al., 2015).

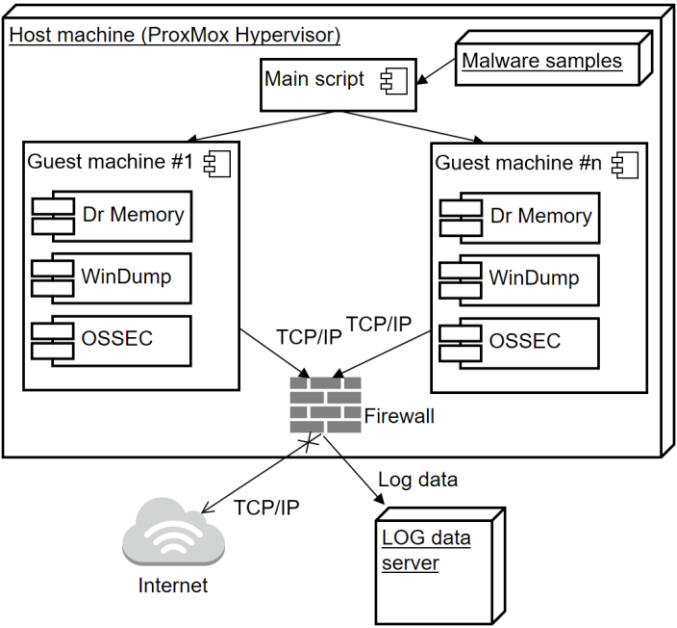


Fig. 2.2. Malware execution components scheme

Open malware collections were used to generate malignant activity on guest machines. Malware execution was conducted on a Windows operating system. For simplicity reasons, during the first step, only malware of executable type was used, in order to minimize dependence on third-party applications (e.g. office suites, utilities, viewers or any other). Malware samples were taken from the freely available database provided by VirusShare (VirusShare.com, 2020) (For this research, VirusShare_00289.zip package, created on 2017-05-07, was used) and theZoo (thezoo.morirt.com, n.d.). VirusShare provides malware packages in the form of password-protected zip files with the usual 'infected' password or any other file types. As a result, every package file type must be analyzed, because there is no file extension provided. Every package can contain various types of malicious files that can target different operating systems: Linux, Windows, Mac, Android and iOS. For that reason, each package must be analyzed, and only Windows OS-executable malicious samples have been selected in our case. VirusShare samples were combined with theZoo malware collection, which holds the most popular and controversial malware samples. theZoo database already contains

password-protected archives with executable files. For that reason, no special preparation procedure was required.

Malware sample preparation is presented in Figure 2.3.

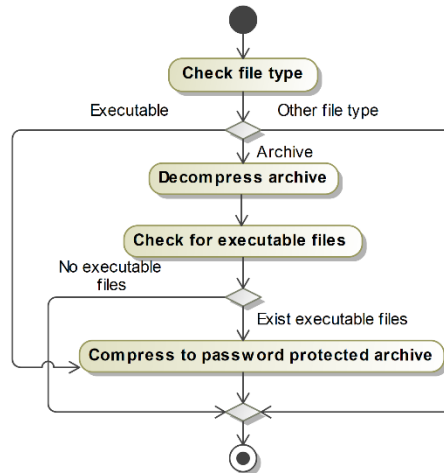


Fig. 2.3. Malware file preparation

Usually, the first byte of a file is holding information about the file type. If that is already a Windows-executable file – a corresponding file extension is added to it, and the file is packed to the archive with a password “infected”. If the analysed file is an archive – it is extracted for further analysis and, if executable files are found, they are added to the password-protected archive. All other files are skipped.

2.3. The Process of Malware Loading to the Guest Machine

Malware transfer to the guest machine was implemented with the help of Proxmox VE, which provided the capability to attach an additional virtual drive and copy the file straight to it. It is not dependent on any other third-party software, and firewall configuration has no impact on file transfer. The number of malware samples that can be executed in parallel, thus influencing the dataset generation speed, depends on the number of running guest machines that are directly related to the available hardware resources.

Our experiments were performed on the HP ProLiant DL 380 G6 server with the following specifications: 2×Xeon E5520 CPU, 8 GB of RAM and 4×146 GB HDD's connected to RAID 5. Six guest machines were running in parallel.

A bash script on the host machine was used to control the guest machine's state (start-up and shutdown) and malicious file transfer to the corresponding virtual drive. Virtual drive preparation for the guest machine also was implemented via bash script. To ensure such method on the newest ProxMox VE – thin provisioning must be turned off. After that, every guest machine drive is represented on a hypervisor as a simple local file. Most importantly, it can be mounted on a hypervisor system and updated with required malware file. Main actions performed by the bash script on the host machine are:

1. Copying guest machines disks from prepared templates;
2. Mounting virtual disk for every guest machine, copying prepared malware, unmounting disk;
3. Starting the guest machine;
4. Pausing script for defined time to provide the malware the possibility to reveal all functionality and features. The default pause time in tests was equal to 30 minutes but can be optimized for generating sets of specific malware types (e.g. longer for botnet and shorter for worm);
5. Stopping the guest machines. The Stop command will halt the machine immediately. The shutdown process is not initiated.

Basically, all steps combine simple commands: copy, machine start and machine stop. However, step number 2 requires a more sophisticated knowledge of virtual disk handling commands.

Guest machine images were also prepared. Each guest machine was running Windows 7 OS and Dr. Memory, OSSEC agent and WinDump. Additionally, there was installed Notepad++ editor and Mozilla Firefox browser for more comfortable support. A malware execution script was added to the Windows task scheduler. Task scheduler provides all required privileges for an unimpeded application/malware startup. It means, that all applications were executed with administrator privileges. Then defined archive file is extracted, malware is executed by a run command for every executable in the extracted folder. The anomaly data gathered (list of modified/accessed files, system calls with related information and network data) was sent to the LOG server for analysis.

All actions required for implementing malware samples execution are presented on Figure 2.4. Malware samples are executed in a batch manner. Every batch has a number of files identical to the number of available guest machines. It can be seen, that host machine waits for the predefined time while a script on guest machine is executing the provided malware sample. This pause is needed to collect anomaly activities in case malware manifests itself after some delay after infecting the machine. No interruptions or errors related to the malware execution

were noticed, which is an advantage against a well-known tool for such task – Cuckoo sandbox. According to Miller et al. – it has stability issues that cause Cuckoo samples results to be inconsistent between runs (Miller et al., 2017).

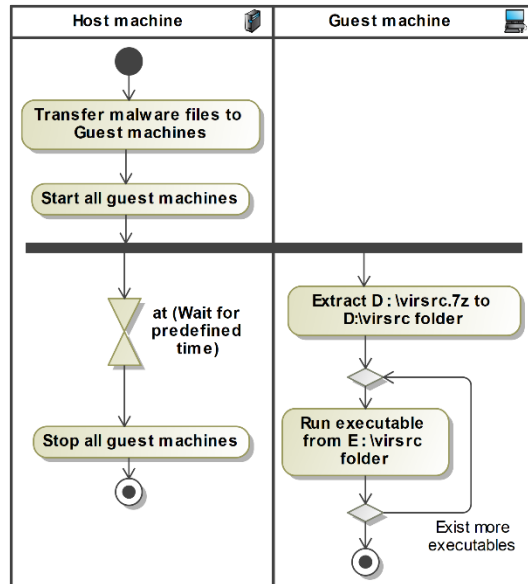


Fig. 2.4. Malware file preparation

2.4. Attack-Caused Windows System Calls Traces Dataset

While developing the dataset (further referred to as Attack-Caused Windows System Calls Traces Dataset or AWSCTD) the following three main objectives were targeted for the dataset generation process:

1. Malware/attack tools have to be publicly available in order to assure easy renewal of dataset in future database renewal and independent verification of results.
2. The dataset should contain the following information:
 - a) All possible information about the system call (function name, passed arguments list and values of them, return value);
 - b) List off changed files affected by malware attack/tools;
 - c) Network traffic generated by malware attack/tools;

3. The dataset should be based on a relatively high number of system calls generated by a wide selection of malware/attack tools.

2.4.1. Dataset Size and Structure

A total of 12110 executable malware samples in the form of password-protected archives were prepared from VirusShare provided packages and used for dataset generation. All samples were tested in a two months period (2017.07–2017.08) and 89.34% (10 820) of them were executed successfully, i.e. the selected malware sample has infected the custom-made test system with Windows OS running on it. Further verification has revealed that not all malware samples acquired could be considered to be 100% malicious. Some of them did not have a proper amount of positive detection rate reported on VirusTotal.com site (VirusTotal, 2020). Because of that, only samples with 15 and more positive detections were selected for the dataset.

Table 2.1. Most common malware categories and families used in dataset generation

Category	Count	Family	Count
AdWare	5139	AdWare.Win32	4655
Trojan	2353	Trojan.Win32	2326
Downloader	853	Downloader.Win32	830
WebToolbar	659	WebToolbar.Win32	654
DangerousObject	137	AdWare.MSIL	412
Trojan-Ransom	101	DangerousObject.Multi	137
Backdoor	79	Trojan-Ransom.Win32	96
RiskTool	55	Backdoor.Win32	75
Trojan-Downloader	45	AdWare.NSIS	71
Trojan-Spy	37	RiskTool.Win32	46
Packed	34	Trojan-Downloader.Win32	44
Virus	17	Packed.Win32	34
Trojan-PSW	16	Trojan-Spy.Win32	34
Trojan-Dropper	15	Downloader.NSIS	19
Trojan-Clicker	5	Trojan.MSIL	18

Distribution of selected malware is presented in Table 2.1 (“DangerousObject” category according to Kaspersky: Malicious software is detected by KL Cloud Technologies. This verdict used for samples that were not classified exactly.). Malware category and family information is based on VirusTotal.com

classification. Only malware detected by at least 15 antivirus vendors was analysed further – this rule has allowed us to select 10 276 malware samples from 10 820 tested.

Collection of system calls traces was performed with the help of drstrace tool, developed by Dr. Memory (DynamoRIO, 2020). It allowed to gather all required system calls information: system call name, passed parameters information (parameters count and values), return values and execution result (success or false). A sample system call is presented on Figure 2.5.

```
NtQueryValueKey
  arg 0: 0x35a (type=HANDLE, size=0x4)
  arg 1: 26/28 "DescriptionID" (type=UNICODE_STRING*, size=0x4)
  arg 2: 0x2 (type=int, size=0x4)
  arg 3: 0x02c5daa0 (type=<struct>*, size=0x4)
  arg 4: 0x90 (type=unsigned int, size=0x4)
  arg 5: 0x02c5da7c (type=unsigned int*, size=0x4)
  failed (error=0xc0000034) =>
  arg 3: <NYI> (type=<struct>*, size=0x4)
  arg 5: 0x02c5da7c => 0x0 (type=unsigned int*, size=0x4)
  retval: 0xc0000034 (type=NTSTATUS, size=0x4)
```

Fig. 2.5. System Call sample from drstrace tool

All system calls were recorded in that format to the log files. A special logs conversion into a better-defined formats (JSON and SQLITE database) was implemented in separate C++ application. A part of code, presenting logs conversion to SQLITE database is shown on Figure 2.6.

A total of 112.56 million system calls traces (table SystemCall) generated by 10276 malware samples were recorded to the database. The total amount of files executed – 17885 (Table MalwareFile). Such an amount of data had a massive impact on the database size – the generated SQLITE file consumes 39.1 GB of storage space. The database contains not only system calls but also metadata about the malware (database scheme can be seen in Figure 2.7).

Part of that information (Malware info) was imported with the help of Academic API provided by Virus Total. The information for every malware record has included:

- Scan engines (antivirus applications) that provided information about the detected threat: e.g. malware type;
- Positive scan results value;
- Web page to malware description page;
- Malware behaviour information:
 1. File system action;

- 2. Network communication;
- 3. Loaded modules (dll files).

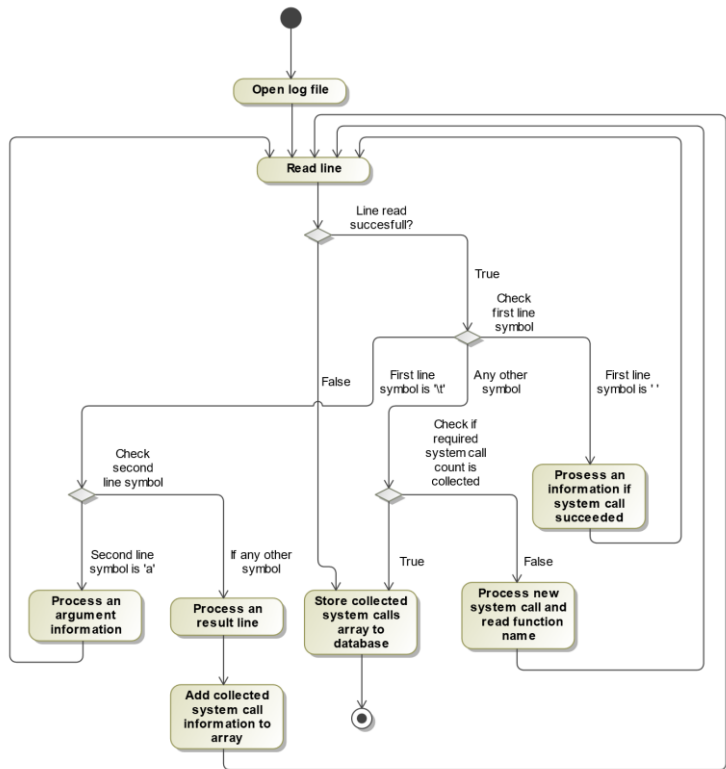


Fig. 2.6. Part of source code of system calls log transfer to the database

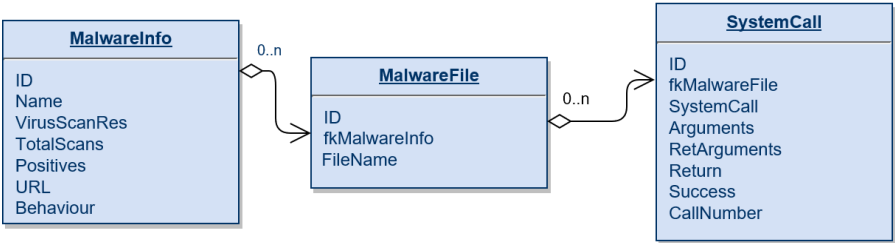


Fig. 2.7. Generated malware activity database scheme

A sample of exported record from the database is provided below in Tables 2.2, 2.3 and 2.4.

Table 2.2. Data sample of Table “Malware Info”

Field	Value
ID	2
Name	000600ee5aedc6e5d4ca946b99f3c924
VirusShare	1
VirusScanRes	"Kaspersky": { "detected": true, "result": "AdWare.Win32.MultiPlug.nnnn", "update": "20171024", "version": "15.0.1.13" },
TotalScans	67
Positives	54
URL	https://www.virustotal.com/file/6abbf5200f267e482b363c4634db9b7213c746ef03cae20ff65da7b8c14d0866/analysis/1508868909/
Behaviour	"runtime-dlls": [{ "file": "advapi32.dll", "success": true }, { "file": "shell32.dll", "success": true }, { "file": "apphelp.dll", "success": true }]

Table 2.3. Data sample of table “MalwareFile”

Field	Value
ID	1
fkMalwareInfo	2
FileName	drstrace.VirusShare_000600ee5aedc6e5d4ca946b99f3c924.exe.02320.0000

Table 2.4. Data sample of table “SystemCalls”

Field	Value
ID	72
fkMalwareFile	1
SystemCall	NtCreateFile
Arguments	["FILE_DIRECTORY_FILE FILE_SYNCHRO- NOUS_IO_NONALERT FILE_OPEN_FOR_BACKUP_IN- TENT FILE_OPEN_REPARSE_POINT (type=named constant, value=0x204021, size=0x4)", "0x002fe874 (type=HANDLE*, size=0x4)", "0x100001 (type=unsigned int, size=0x4)", "len=0x18, root=0x0, name=90/538 \\\"\\??\\C:\\Us- ers\\elsa\\AppData\\Local\\Temp\\2C9792C0\\", att=0x40, sd=0x00000000, sqos=0x00000000 (type=OBJECT_ATTRIB- UTES*, size=0x4)", "0x002fe864 (type=IO_STATUS_BLOCK*, size=0x4)", "<null> (type=LARGE_INTEGER*, size=0x4)", "FILE_ATTRIBUTE_NORMAL (type=named constant, value=0x80, size=0x4)", "FILE_SHARE_READ FILE_SHARE_WRITE (type=named constant, value=0x3, size=0x4)", "FILE_CREATE (type=named constant, value=0x2, size=0x4)", "<null> (type=<struct>*, size=0x4)", "0x0 (type=unsigned int, size=0x4)"]
RetArguments	["", "0x002fe874 => 0xb0 (type=HANDLE*, size=0x4)", "", "", "status=0x0, info=0x2 (type=IO_STATUS_BLOCK*, size=0x4)", "", "", "", "", "", "", "", ""]
Return	0x0 (type=NTSTATUS, size=0x4)
Success	1
CallNumber	72

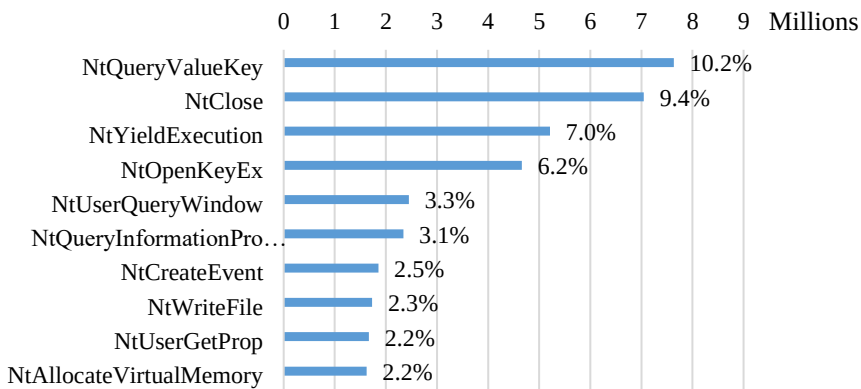
Table 2.5. ADFA-WA dataset comparison to AWSCTD dataset

Dataset	Executed malware samples	Collected System Calls count
ADFA-WA	15	6636
AWSCTD	10 276	112.56 million

As it can be seen from Table 2.5, AWSCTD exceeds the only currently available dataset by the number of tested malware and generated system calls count by order.

2.4.2. Dataset Characteristics

According to the (Miao et al., 2006) there are about 949 native calls (284 key APIs from Ntdll.dll and 665 less important from Ntoskrnl.dll) in the already discontinued Windows XP operating system. In our tests performed on the basis of Windows 7 OS, 645 distinct system calls were captured. The most commonly used system calls are presented on Figure 2.8. The dominating part of calls generated by the malware was related to registry querying. The next dominating group of calls was implementing the file processing functions (reading and writing). The system calls success rate parameter obtained was very high – 99% of all executed calls have returned the desired result. Dr. Memory was used to evaluate the success rate. The calls with the highest success rate were NtQueryValueKey and NtOpenKeyEx. The lowest rate was demonstrated by NtYieldExecution (this function stops execution of thread calling and switches to any other currently running thread) and NtCallbackReturn (this function finishes execution of User-Mode callback).

**Fig. 2.8.** Most used system calls by the malware

As stated earlier, some characteristics of malware behaviour (mainly related to file access) were obtained from Virus Total. The most frequently requested files are listed in Table 2.6. In this category, not only critical Windows system files can be noticed, but also various services: such as driver responsible for maintaining persistent drive letters and volume names (MountPointManager), or remote procedure calling (lsarpc) used by the applications based on the client-server architecture.

Table 2.6. The most frequently requested files

File path	Count	Percentage
\\.\PIPE\lsarpc	1080	0.3%
C:\WINDOWS\Registration\R0000000000007.clb	766	0.2%
\\.\Ip	681	0.2%
\\.\MountPointManager	652	0.2%
c:\autoexec.bat	611	0.1%
C:\WINDOWS\system32\shdocvw.dll	326	0.1%
C:\WINDOWS\system32\msi.dll	248	0.1%
C:\WINDOWS\system32\stdole2.tlb	221	0.1%
C:\WINDOWS\WindowsShell.manifest	218	0.1%
C:\Program Files\Internet Explorer\iexplore.exe	210	0.1%

Table 2.7. NtCreateFile parameter “CreateOptions” most frequently used combinations

Parameter combinations	Count
FILE_SEQUENTIAL_ONLY FILE_NON_DIRECTORY_FILE	8434
FILE_SEQUENTIAL_ONLY FILE_NON_DIRECTORY_FILE FILE_OPEN_REPARSE_POINT	8431
FILE_NON_DIRECTORY_FILE FILE_OPEN_NO_RECALL	374
FILE_SYNCHRONOUS_IO_NONALERT FILE_NON_DIRECTORY_FILE	
FILE_OPEN_NO_RECALL	221
FILE_SYNCHRONOUS_IO_NONALERT	129
FILE_SEQUENTIAL_ONLY FILE_SYNCHRONOUS_IO_NONALERT	
FILE_NON_DIRECTORY_FILE FILE_OPEN_NO_RECALL	10
FILE_OPEN_FOR_BACKUP_INTENT	6
FILE_SEQUENTIAL_ONLY FILE_SYNCHRONOUS_IO_NONALERT	4

AWSTCD dataset provides additional system call information: a list of parameters, which were not included in previous publicly available datasets. These parameters can provide additional information on malware behaviour while performing ML methods training. A sample of parameters combinations, together with statistics of the NtCreateFile system call, are provided in Table 2.7 (parameter CreateOptions) and Table 2.8 (parameter ShareAccess).

Table 2.8. NtCreateFile parameter “ShareAccess” most frequently used combinations

Parameter combinations	Count
FILE_SHARE_READ	340 030
FILE_SHARE_READ FILE_SHARE_WRITE	233 424
FILE_SHARE_READ FILE_SHARE_WRITE FILE_SHARE_DELETE	92 609
FILE_SHARE_READ FILE_SHARE_DELETE	85 237
0x0	27 297
FILE_SHARE_WRITE	914
0xffffffff90	3

The parameter combinations used with system calls usually vary for legal applications and malware, therefore this information can be used as one of the distinguishing characteristics of malware behaviour. The dataset includes parameter combinations in the same form as presented for all system calls included in the dataset.

Despite the fact that dataset is extensive and contains comprehensive system calls information compared to the other datasets in the field, it has several limitations that should be addressed. The variety of malware samples could be wider for some classes such as ransomware (currently there are 96 samples of Trojan-Ransom.Win32 family). Secondly, the number of records by benign programs could be also expanded to make the dataset more balanced.

2.5. AWSCTD Dataset Balancing

The dataset balancing methods are well known for the research community. They are covering a problem of imbalanced classes: when one class has a lot more samples (majority) than other (minority) (Haixiang et al., 2017). This situation can lead to improper classification. For example, in fraud detection, only a few cases are fraud cases, and detection of them is a complicated task as the algorithm has only a few samples to learn of (Sanz et al., 2015).

Oversampling techniques are often used to battle data samples inequality. One of them is a Synthetic minority over-sampling technique (SMOTE). It works by creating synthetic minority class examples (N. V. Chawla et al., 2002). The results of N. V. Chawla et al. paper shown that the SMOTE approach can improve the accuracy of classifiers for a minority class and the combination of SMOTE and under-sampling performs better than plain under-sampling.

Although the oversampling technique could improve classification results – it has been decided not to balance AWSCTD dataset (in later chapter new minority class of benign applications has been added). The main reason – it is essential in intrusion and malware detection task to have more samples of malignant behaviour and most importantly, they must be real and not generated synthetically.

2.6. Methods for Minimising Overfitting Problem

According to Oxford dictionary, in statistics, overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. Overfitting is one of the major problems in DL research field because it was already mentioned next to first studies and is battled since then (Connor & Atlas, 1991).

Table 2.9. The summary of overfitting minimization methods usage in the practical thesis part

Method	Usage description
Cross-validation	10-fold cross-validation is used in section 3 as the default method proposed by Weka. 5-fold cross-validation with the shuffle parameter as <i>true</i> is used in section 4. 10-fold cross-validation was consuming a significant amount of time when DL models were used. For such reason, a 5-fold variant was decided to use.
Dropout	The dropout technique was used in section 4 for certain RNN based FCN and AWSCTD models. An aggressive dropout coefficient of 0.8 was used.
Early stopping	Early stopping when the accuracy of the trained model is not changing for six epochs was used in section 4 for all DL models training.

Several methods evolved since then to overcome such disadvantage: cross-validation, dropout and early stopping. All three methods have been used in this thesis experiments with DL methods. A 5-fold Cross-validation technique applied for training and testing the data splitting procedure. Early stopping adapted to stop

training at a certain point when accuracy is not changing significantly. And finally – dropout used in DL models as it is recognized as one of the most effective methods to overcome overfitting. Dropout technique was found to improve the performance of neural nets in a wide variety of application domains including object classification, digit recognition, speech recognition, document classification and analysis of computational biology data. This suggests that dropout is a general technique and is not specific to any domain (Srivastava et al., 2014). The overfitting minimizations methods usage in this thesis practical part is concluded in Table 2.9.

2.7. Classification Methods Evaluation Criteria

To evaluate the efficiency of ML and DL models for the dataset, the following metrics are used as standard: accuracy, confusion matrix (CM), precision, recall, F-score, FPR, False Negative Rate (FNR), classification error, Receiver Operating Characteristic (ROC) curve, Mathews Correlation Coefficient (MCC), training and testing times. The accuracy metric is calculated for both benign and malware data classes, while the precision, recall, F-score, MCC, FPR, FNR and Error rate are calculated only for the malware class and represent the model capability to perform the intrusion/malware detection task.

Typically, to divide dataset into training and testing sets - *Cross-Validation* method is used. K-fold cross-validation is widely used as criteria to evaluate datasets. Although k can be any integer number, 5-fold or 10-fold cross-validations are often applied. In 5-fold cross-validation, the original dataset is divided into 5 partitions of the same size, four partitions are used for training, and one partition is used for testing. Then another four partitions and the leftover one partition will be used for training and testing. This cross-validation approach will repeat for 5 times with different training and testing data, and 5 results will be generated and averaged to get the final result. The K-fold cross-validation method can provide a more precise estimation of the whole dataset (Dua & Du, 2016).

The *accuracy* is calculated as follows (explanation is based on research related to the thesis): the method of machine learning is trained with a portion of the dataset (80%), while another portion (remaining 20%) of the dataset is used for testing with the trained model, i.e. data used for testing had not been used for training. Therefore, the percentage of correctly classified records is defined as accuracy (Čeponis & Goranin, 2019). This method is repeated in five K-Folds, and average accuracy based on Keras accuracy metric (Chollet & others, 2015) is used, representing an average classification capability for all classes.

The *Confusion Matrix*: given a classifier and a set of instances (the test set), a two-by-two CM (also called a contingency table) can be constructed representing the dispositions of the set of instances. This matrix forms the basis for many common metrics (Fawcett, 2005). CM used for intrusion classification can be seen in Figure 2.9.

		Predicted label	
		Benign	Malware (Intrusion)
True label	Benign	TN	FP
	Malware (Intrusion)	FN	TP

Fig. 2.9. CM of the intrusion classification

The *precision* (also called *detection rate*) is the fraction of relevant instances among the retrieved instances (Fawcett, 2005). It means that when the model predicts a positive value, then what are the odds that the model has made a correct prediction. The precision is calculated as follows:

$$\text{Precision} = \frac{TP}{TP+FP}. \quad (2.1)$$

The *recall* (also known as *sensitivity*) is used to measure the fraction of positive patterns that are correctly classified (Fawcett, 2005). In other words - recall is a measure that tells us how great our model is when all the actual values are positive. The recall is calculated as follows:

$$\text{Recall} = \frac{TP}{FN+TP}. \quad (2.2)$$

The *F-score* is a metric that combines *recall* and *precision* by taking their harmonic mean (Fawcett, 2005). F-score is calculated as follows:

$$\text{F score} = \frac{2(\text{Recall Precision})}{\text{Recall}+\text{Precision}}. \quad (2.3)$$

The *MCC* is not a biased metric (as precision or recall) and is flexible enough to work properly even with highly unbalanced data. Main properties of MCC are (Boughorbel et al., 2017):

- The MCC can be calculated using the CM.

- The calculation of the MCC metric uses the four quantities (TP, TN, FP and FN), which gives a better summary of the performance of classification algorithms.
- The MCC is not defined if any of the quantities TP + FN, TP + FP, TN + FP, or TN + FN is zero.
- MCC takes values in the interval $[-1, 1]$, with 1 showing a complete agreement, -1 a complete disagreement, and 0 show that the prediction was uncorrelated with the ground truth.

The MCC value is calculated as follows:

$$MCC = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}. \quad (2.4)$$

The *False Positive Rate* (FPR) (also called False Alarm Rate) is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events (Axelsson, 2000). This parameter in intrusion detection is important because it shows how wrongly model has classified an action as non-intrusion or valid. The aim is to have as small FPR as possible to prevent the security personnel from false alarms. FPR is calculated as follows:

$$FPR = \frac{FP}{FP+TN}. \quad (2.5)$$

In the same way the *False Negative Rate* (FNR) shows the ratio of not detected intrusion attempts and should also be minimized (Joo et al., 2003). FNR is calculated as follows:

$$FNR = \frac{FN}{FN+TP}. \quad (2.6)$$

The *Classification Error* (or Error Rate) tells how often the model makes a wrong prediction (Fawcett, 2005). The formula for classification error is the following:

$$\text{Classification error} = \frac{FP+FN}{TP+TN+FP+FN}. \quad (2.7)$$

The *ROC curve* has FPR on the x-axis and TPR on the y-axis. In ROC curve the True Positive Rate (TPR) is plotted as a function of the FPR for different cut-off points. Each point on the ROC curve represents an FPR and TPR pair corresponding to a certain decision threshold. As the threshold for classification is varied, a different point on the ROC is selected with different False Alarm Rate (FAR) and different TPR. A test with perfect discrimination (no overlap in the two distributions) has a ROC curve that passes through the upper left corner (100% sensitivity, 100% specificity) (Davis & Goadrich, 2006). The ROC Curve is shown in Figure 2.10.

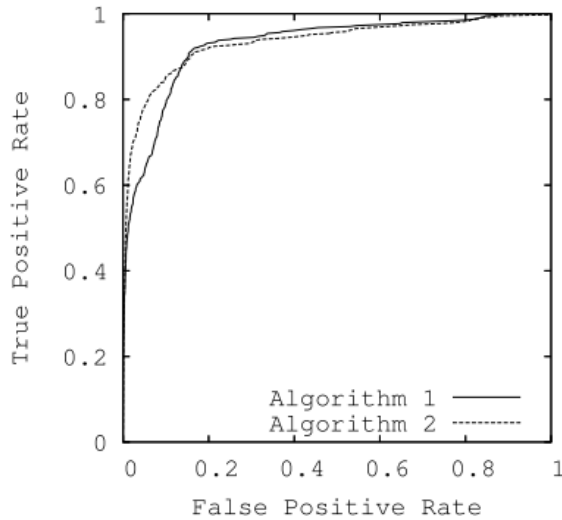


Fig. 2.10. Sample of two ROC curves compared (Davis & Goadrich, 2006)

The *training and testing times* usually demonstrate the time needed to train a model and identify the intrusion/malware, respectively. For intrusion detection – the one-sample test time (decision does it is an intrusion/malware or valid application) is more significant as the ML models are not renewed/trained very often.

2.8. Conclusions of Chapter 2

1. The method for host-level anomaly dataset generation was proposed. The decision was made to utilise virtualization technology in order to isolate the potentially dangerous dataset generation environment. The proposed method is based on malware execution in a sterile, isolated virtual machine environment with further anomaly activity collection and data representation in an SQLite database format. While performing dataset generation there were no cases of malware escaping from isolated environment, which proves that use virtualization technologies can be considered as a reliable solution for dataset generation when data-generating systems should be securely isolated;
2. The method was implemented and tested only with free or open-source tools and freely available malware samples. The tests performed have proved the method stability and method suitability for host-level anomaly

dataset generation. Automated anomaly generation allows flexible training data-set expansion, response to the new attack types and generation of specific on-demand datasets;

3. AWSCTD was generated for 10 276 malware samples. The dataset size exceeds the size of previously known datasets by order and includes a much more comprehensive representation of malware types and system calls. AWSCTD has an additional advantage against existing datasets because of parameters (system call arguments list and return value) that allow more in-depth HIDS training;
4. An expansion of the generated dataset is being planned for creating a more comprehensive host-level anomalies dataset for HIDS training. The expansion is planned via the inclusion of non-executive type malware samples, non-malware attacks and optimizing the pause interval for better feature assembly of delayed malware activities.

Evaluation of Machine Learning Methods Application for Anomaly-Based Intrusion Detection and Classification

This Chapter presents an experimental investigation of ML methods application for malware classification by using system calls. The newly generated AWSCTD dataset has been used as the source. Seven ML methods have been selected to apply: Bayes Net, Naïve Bayes Net, Support Vector Machines, Neural Networks, Decision Table, Decision Tree J48 (C4.5), Logistic Model Tree (LMT). The chapter also includes the selection of the most appropriated evaluation criteria, description of the testbed configuration, actions performed for feature pre-processing. The chapter concludes with an evaluation of the tested ML methods and results achieved.

The experiment results were published in the international journal and presented at the international conference (Goranin & Čeponis, 2018).

3.1. Application of AWSCTD for the Malware Classification

The total of 5 malware families was selected from the dataset for our experiment. The main selection criteria – family should have more than 100 samples of unique family representatives, and the corresponding system calls sequences. Kaspersky provided family descriptor was used. Table 3.1. shows the number of unique malware samples in each family (“DangerousObject” category according to the Kaspersky: Malicious software is detected by KL Cloud Technologies. This verdict is used for samples that were not classified exactly.).

Table 3.1. The number of each malware label

Label	Count
Trojan	1755
AdWare	4333
WebToolbar	618
Downloader	710
DangerousObject	105
Total:	7521

The different size of families represented helps us to evaluate the influence of the size of an initial dataset on the accuracy of classification.

3.2. Experimental Testbed Configuration and Selected Criteria

A WEKA 3.8 software has been used for the classification. A relatively simple hardware configuration was used to support the above-mentioned software:

- CPU: Intel(R) Core (TM) i7-7500U 2.70 GHz (2 Cores, 4 Threads);
- RAM: 8GB (DDR3);
- OS: Windows 10 PRO.

To evaluate the efficiency of all models the following metrics were defined: accuracy, confusion matrix (CM), ROC curve, and training and testing times. Those criteria were proposed as the most important by the used software itself and are sufficient to primary analysis of newly created dataset.

3.3. Feature Pre-Processing and Selected ML Methods

We are stating that it is enough to test the first batch of application system calls to determine its family. For that reason, the first 10, 20, 40, 60, 100, 200, and 400 system calls were used to generate the training and testing sets. The lengths of system calls sequences were determined experimentally. At first short sequences of ten system calls were used, then it was extended by 10–20 calls. Later it was extended by 100–200 system calls to achieve better classification accuracy. The sample for 10 first system calls can be seen in Figure. 3.1. Every system call is represented by the unique number, and the sequence of these numbers forms the feature vector used for training.

```
@RELATION SystemCalls0010
@ATTRIBUTE SystemCall0 NUMERIC
@ATTRIBUTE SystemCall1 NUMERIC
@ATTRIBUTE SystemCall2 NUMERIC
@ATTRIBUTE SystemCall3 NUMERIC
@ATTRIBUTE SystemCall4 NUMERIC
@ATTRIBUTE SystemCall5 NUMERIC
@ATTRIBUTE SystemCall6 NUMERIC
@ATTRIBUTE SystemCall7 NUMERIC
@ATTRIBUTE SystemCall8 NUMERIC
@ATTRIBUTE SystemCall9 NUMERIC
@ATTRIBUTE class {Trojan,AdWare,WebToolbar,Downloader,
DangerousObject}

@DATA
7,10,10,9,3,3,3,9,10,10,AdWare
7,10,10,9,3,3,3,9,10,10,AdWare
20,20,20,10,10,9,9,9,18,11,Trojan
7,10,10,9,9,9,3,3,3,9,AdWare
9,18,21,22,9,9,26,9,18,23,AdWare
29,19,2,13,31,1,2,28,28,33,Trojan
7,10,10,9,3,3,3,9,10,10,AdWare
```

Fig. 3.1. WEKA data file fragment for the first ten system calls

A WEKA 3.8 data mining software was used to test various classification methods (Witten et al., 2005). The ability to use independent open source and customizable software package was the main reason for such selection. None of the WEKA-provided default method values were changed or tinkered to achieve

better results – all methods were used with default hyperparameter values. Different type of Weka provided ML methods were selected to test the accuracy of malware type classification. Methods with training values can be seen in Table 3.2.

Table 3.2. Used ML methods with training parameters

Method	Training parameters
Bayes Net	searchAlgorithm: K2 maxNrOfParents: 1 scoreType: BAYES estimator: SimpleEstimator alpha: 0.5 useADTree: False
Naïve Bayes Net	No classifier related parameters.
Support Vector Machines	seed: 1 nu: 0.5 cacheSize: 40 cost: 1.0 eps: 0.001 degree: 3 kernelType: 2 gamma and coef0: 0.0 SVMType: 1 loss: 0.1
Neural Networks	learningRate: 0.3 momentum: 0.2 trainingTime: 500 validationSetSize and seed: 0 validationTreshold: -20 hiddenLayers: a ((attribs + classes) / 2)
Decision Table	crossVal: 2 search: BestFirst lookupCacheSize: 1 searchTermination: 5
J48 (C4.5)	confidenceFactor: 0.25 minNumObj: 2
LMT (Logistic Model Tree)	minNumInstances: 15 numBoostingIterations: -1 weightTrimBeta: 0.0

3.4. Results of Malware Classification

Classifiers were trained and tested with a 10-fold cross-validation technique. Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model and a test set to evaluate it. Our selected out of the box ML methods allowed to achieve results suitable for practical application. Almost all of them have achieved more than 80% off accuracy for the dataset, with the best result of 92.4 by SVM. The percentage of correctly classified malware samples by different methods can be seen in Table 3.3.

Table 3.3. Correctly classified percentage of instances

Feat. count	Bayes Net	Naïve Bayes	SVM	NN	Dec. Table	J48	LMT
10	87.5	61.6	89.4	87.2	88.1	88.5	89.1
20	73.8	60.9	89.4	87.8	88.0	88.8	89.2
40	74.3	61.7	91.6	88.5	89.4	90.7	91.0
60	75.0	66.1	91.8	90.2	89.3	90.9	91.2
100	75.4	61.8	92.4	91.4	90.0	92.1	92.1
200	76.1	68.2	89.8	90.3	90.0	91.9	91.8
400	75.9	69.3	87.5	84.8	89.7	92.1	92.1

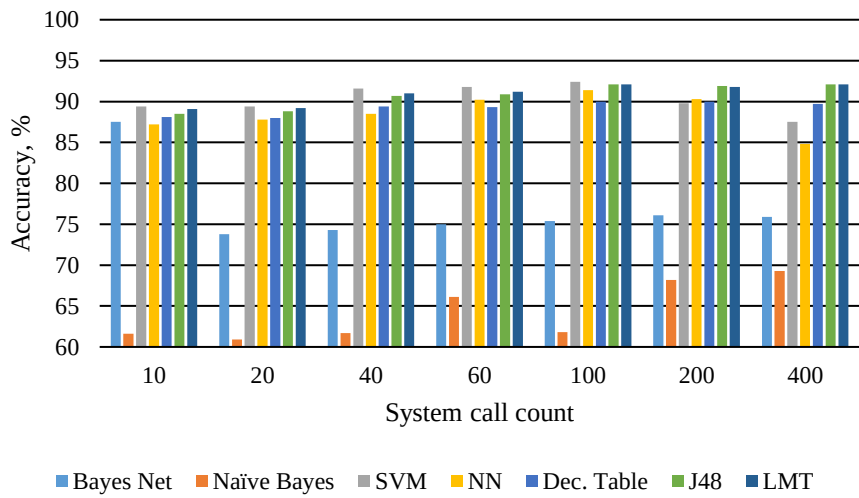


Fig. 3.2. Accuracy results of ML methods

The best results of classification are represented by the green colour in the Table 3.3. The worst values are in red. Graphical representation of results can be seen in Figure 3.2.

Worst results were obtained by using Bayes classification family: Bayes Net and Naïve Bayes Net. Naïve classification method produces the worst of all results: achieving only 60.9% of correct classification with a mean absolute error (MAE) of 0.17. SVM has achieved the best result of 92.4% with the 100 system call long feature vector (MAE of 0.03). It is not surprising as SVM's are resistant to overfitting and always returning global optimum in comparison with other classifiers such as NN. The high results of SVM indicate that training data is easily separable (because it operates better with such data), and that demonstrates the high quality of the collected dataset. It is clear that 100 features vector is the most optimal subset to achieve more than 90% accuracy. Decision trees algorithms have also produced remarkable results for feature vector length of 100 and 400: it was even better than NN and had an accuracy of 92.1% (both methods produced equal MAE of 0.05). That is expected since already in 2008 the C4.5 decision tree algorithm was one of the best ten choices for the data mining (Wu et al., 2008).

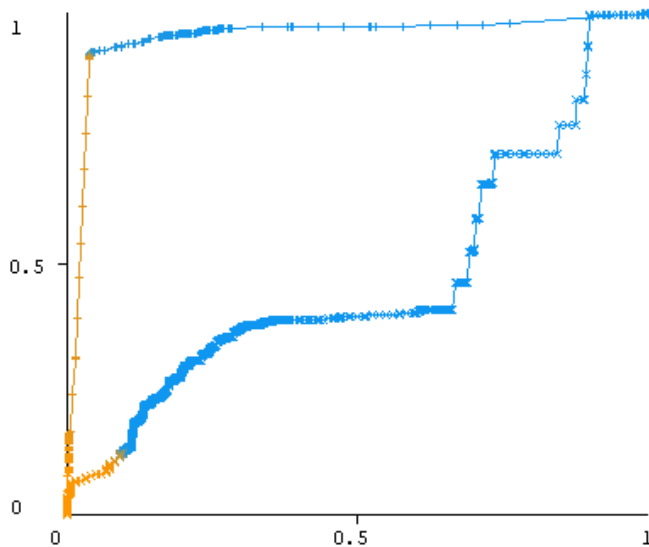


Fig. 3.3. ROC curve of J48 (+) and Naive Bayes (x) classifiers. X-axis – False Positive Rate. Y-axis – True Positive Rate

Classification performance of J48 vs. Naïve Bayes can be seen on Figure 3.3 J48 area under ROC (AUC) (0.9486) is bigger than Naïve Bayes (0.4664). AUC of Naïve Bayes indicates that it produces worse results than random classification (Fawcett, 2005) which produces AUC of 0.5.

Obviously, the best classification results were obtained for malware families that had the biggest number of the samples. In this specific dataset that were Trojans and AdWare (1755 and 4333 samples, respectively). Family “Dangerous object”, a family with only 105 samples of in fact different types, has shown the lowest detection rate. SVM Confusion Matrix for the 100 features vector can be seen in Figure 3.4.

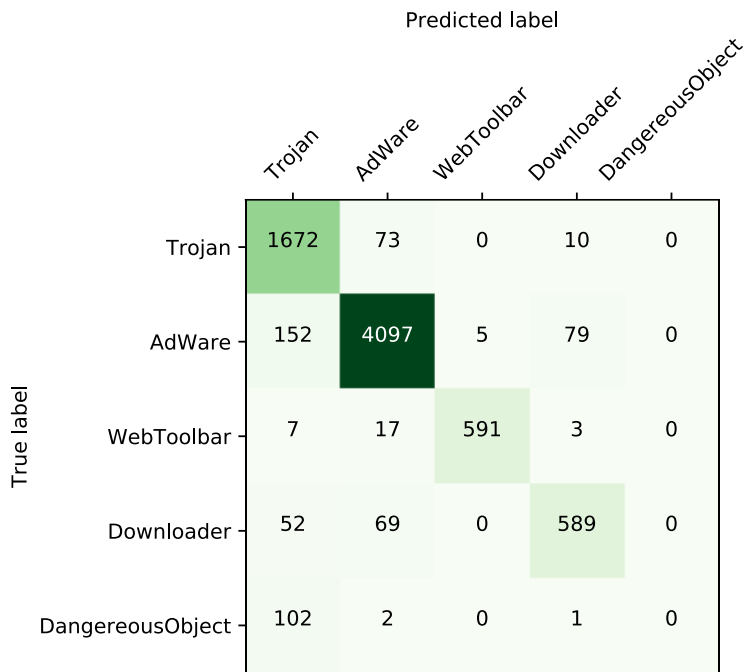


Fig. 3.4. SVM Confusion Matrix for the 100 features vector

When comparing training and testing execution times, decision trees have shown the best results. Both training and testing values are among the shortest. SVM and NN show the worst outcome, even though they produce only 1.3% and 0.3% better performance compared to J48 regarding accuracy. On the contrary, NN is 24 times faster than SVM since they are trained, and the testing stage is

activated. Weka produced elapsed time for training and testing can be seen in Table 3.4 and Table 3.5. A graphical representation of 100 foremost system calls training and classification times can be seen in Figure 3.5 and Figure 3.6. These figures clearly show that training of NN and LMT models takes longer than any other model and its correlating with accuracy results. Contrarily, testing times show that SVM displayed as the most time-consuming model, which is also supported by the fact that it returns best classification accuracy result.

Table 3.4. Weka produced Elapsed_Time_training values (seconds)

Feat. count	Bayes Net	Naïve Bayes	SVM	NN	Dec. Table	J48	LMT
10	0.05	0.01	1.3	8.4	0.4	0.1	14.2
20	0.07	0.01	1.9	17.2	0.9	0.2	21.1
40	0.09	0.02	3.8	46.3	2.8	0.4	40.2
60	0.29	0.04	4.8	90.4	4.2	0.8	69.2
100	0.31	0.06	7.5	222.4	5.5	1.4	106.6
200	0.93	0.12	16.8	938.3	16.0	3.2	301.0
400	3.61	0.28	40.6	4410.1	33.7	9.1	726.0

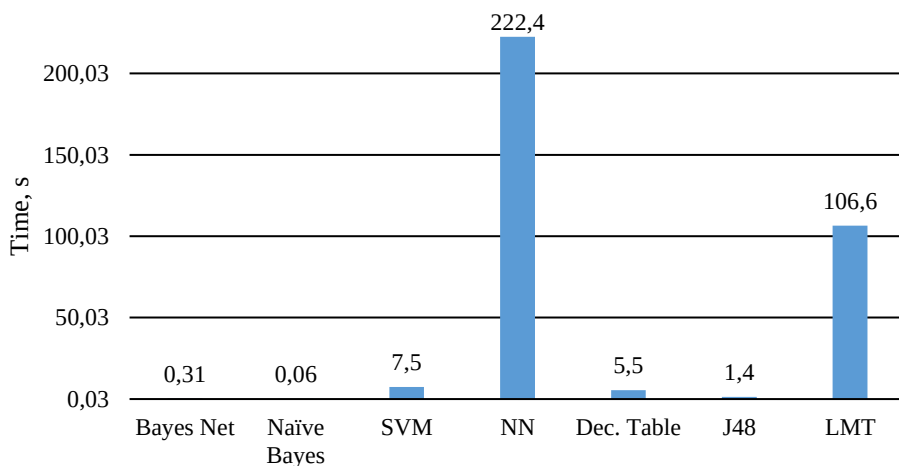


Fig. 3.5. The training time of ML methods for 100 system calls

Table 3.5. Weka produced Elapsed_Time_testing values (seconds)

Feat. count	Bayes Net	Naïve Bayes	SVM	NN	Dec. Table	J48	LMT
10	0.003	0.009	0.130	0.002	0.002	0.005	0.003
20	0.002	0.017	0.184	0.002	0.003	0.000	0.002
40	0.003	0.031	0.309	0.009	0.000	0.002	0.003
60	0.011	0.041	0.396	0.009	0.002	0.003	0.000
100	0.019	0.078	0.586	0.024	0.000	0.000	0.003
200	0.045	0.154	1.303	0.094	0.002	0.000	0.006
400	0.100	0.454	3.098	0.413	0.005	0.000	0.016

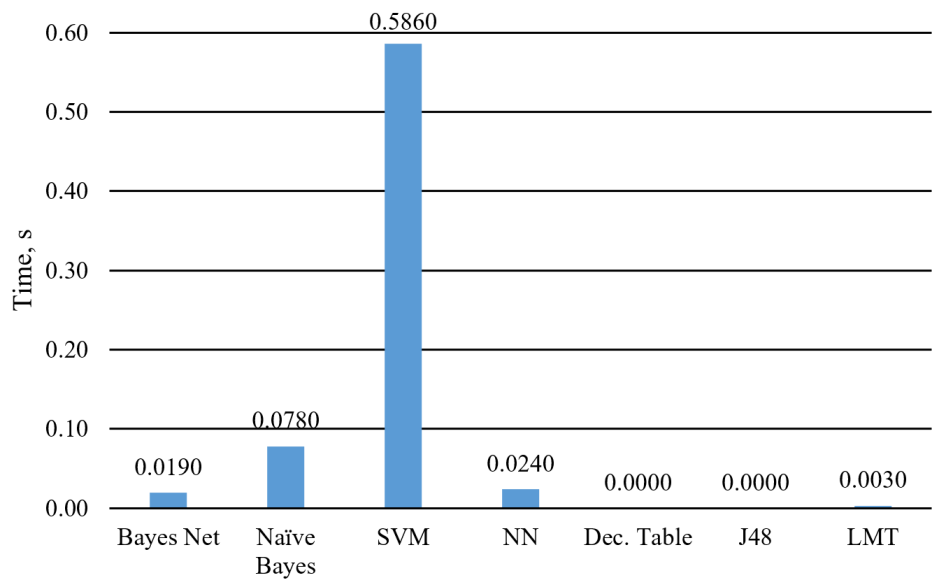


Fig. 3.6. Classification time of ML methods for 100 system calls

A classification performance, FPR, PNR and classification error metrics for the foremost 100 system calls classification have been calculated and are presented in Table 3.6. The results affirm the conclusion that SVM based model is somewhat the best performing model in comparison with other ML models as it produces the highest precision (0.732 when the average is 0.690) and F-1 (0.734 when the average is 0.719) scores. Also, it shows the lowest FPR (2.4% when the average is 4.3%) and classification error (0.076 when the average is 0.15) values in comparison with other ML models in mind that MAE of SVM for 100 system

calls was 0.03. All other models produced higher MAE values: Bayes Net – 0.10; Naïve Bayes – 0.15; NN – 0.04; Decision Table – 0.08; J48 – 0.05; LMT – 0.05. There is a 95% likelihood that the average confidence interval [0.057, 0.095] covers the true classification error of the best performing SVM-based model on unseen data.

Table 3.6. Classification performance, FPR, FNR and classification error metrics of ML models for 100 system calls

Metric	Bayes Net	Naïve Bayes	SVM	NN	Dec. Table	J48	LMT
Precision	0.705	0.509	0.732	0.721	0.719	0.722	0.724
Recall	0.779	0.633	0.737	0.720	0.695	0.734	0.735
F-1	0.622	0.491	0.734	0.720	0.706	0.728	0.729
FPR	0.057	0.099	0.024	0.029	0.037	0.025	0.026
FNR	0.222	0.367	0.263	0.281	0.305	0.266	0.265
Error rate	0.246	0.383	0.076	0.085	0.101	0.079	0.080

All tests were performed on a system with an Intel i5-4670 3.40 GHz CPU. In comparison with the (Kolosnjaji et al., 2016) used TITAN X GPU, it is a much slower device in the means of computation power. Also, we were using ML methods without any parameters optimization and much smaller features vectors for the training and testing, and, all system calls were used for the feature vector generation (authors of (Kolosnjaji et al., 2016) have not included repetitive system calls from their dataset) which provide more information for the ML methods. All these differences let us achieve results that can be also used practically. That leads to the conclusion, that Deep Learning methods are an overhead for a task of this type – correctly classified instanced are over 90% and training time is more suited for the domain that generates new samples every day (model proposed by (Kolosnjaji et al., 2016) has a training time that ranges from three to ten hours).

3.5. Conclusions of Chapter 3

1. The newly generated dataset AWSCTD of Windows OS system calls was used to show the applicability of malware generated systems calls sequences for IDS reported alerts prioritization use. IDS alert prioritization can be seen as an important part of anomaly-based intrusion detection. It

allows minimization of human resources and allows concentrating attention on the most dangerous attacks vectors;

2. Seven types of ML methods were applied to classify malware by family. The results have shown that this approach could be used in practical applications – the best detection rate of 92.4 was obtained by using SVM with the 100 features vector with MAE of 0.03. Comparable results were achieved using decision tree algorithms: J48 and LMT. They generated 92.1% classification results (MAE of 0.05). Still, it can be concluded, that J48 is better for the task specified since it has also shown the best training and testing times;
3. Despite of achieved relatively high accuracy rates reaching 90–92% (suitable for some practical applications when additional analysis is performed by external human experts) and relatively low training time needed for ML methods, further research on DL methods applicability is needed in order to minimize the false-negative rate.

Evaluation of Deep Learning Methods Application for Anomaly-Based Intrusion Detection and Classification

This Chapter presents an experimental investigation of DL methods application for malware classification by using system calls. The AWSCTD dataset has been used as the source. The Chapter will have two main sections: in the first one, vanilla DL models will be tested, and in the second section, the more advanced DL model configurations will be used. The chapter also includes a description of AWSCTD expansion with system calls by benign system calls, the selection of the most appropriated evaluation criteria, description of the testbed configuration, actions performed for feature pre-processing, motivation for selecting DL models and their configuration parameters. The chapter concludes with a comparative analysis of vanilla and advanced DL models results, and the most appropriate method for malware and intrusion detection on the host level is proposed.

The results of the following experiment were accepted published in international journals (Čeponis & Goranin, 2020b; Goranin & Čeponis, 2018).

4.1. Motivation of Dataset and DL Models Selection

The increasing demand for anomaly-based detection on the host level stimulates the research on ML and DL application for the host-level intrusion and malware detection. Some recent research, utilizing relatively vanilla DL methods, has demonstrated good anomaly-based detection results that already have practical applicability due to low FPR and their ability to detect zero-day attacks potentially. However, it was decided to evaluate the applicability of more complex DL methods that typically provide better results in natural language processing and image recognition tasks. Motivation for choosing different model architectures was led by intention to test as many perspective architectures as possible. Perspectives of models were evaluated based on review of their application in similar research areas.

For the experiment, AWSCTD dataset containing system calls sequences from Windows OS was used. It was generated using publicly available malware files from Virus Share and publicly available information about any malware found from Virus Total. Later, the collected database was updated with the additional information provided by the Virus Total, which included scan results and behavioural information.

Only one dataset was selected for the tests since the only comparable dataset to AWSCTD is ADFA-IDS, but performing tests on both of them was not practical because of the following reasons:

- Test on ADFA-IDS with simple ML methods performed by other authors (Xie, Hu, Yu, et al., 2014) demonstrated poorer results compared to the use of AWSCTD in Chapter 3;
- The data structure of ADFA-IDS is somewhat similar to the AWSCTD, but AWSCTD wins by the amount of data (malware samples and the number of system calls for every malware sample);
- Since the number of system calls in the ADFA-IDS is typically smaller than in AWSCTD, it would be impossible to compare the results for longer system call sequences (> 600) that typically produce better intrusion/malware detection results.

For experiments described in this chapter, AWSCTD dataset was appended with 16.3 million system calls generated by a set of 3145 benign applications (samples were taken from Virus Share and carefully filtered to contain only samples with zero detection rate). The system call collection method for the benign application was the same as for malware system call collection described in Chapter 2. This was done to train CNNs and RNNs for malicious/benign activity classification. It is expected that the number of benign applications with related system calls will increase in the future.

The disproportion of system calls versus the number of applications in case of malicious (16.3/3145) and benign programs (112.56/12110) can be explained by the fact of more “aggressive” and “active” malware behaviour compared to legal applications.

System Calls sequences used for experiments, as well as source code used to train and test DL models, can be found on the GitHub repository (Čeponis & Goranin, 2020a).

4.2. Feature Processing

The data generated by the malware and benign samples were stored in an SQLite database. The system calls sequences were stored in the format provided by Dr. Memory DrSTrace tool (see Figure 4.1). To evaluate the influence of a number of system calls on classification/detection rate, eight files in csv format were generated with 10, 20, 40, 60, 80, 100, 200, 400, 600, 800 and 1000 of the first system calls in every line in a file respectively (see Table 4.3).

```
NtQueryValueKey
  arg 0: 0x158 (type=HANDLE, size=0x4)
  arg 1: 16/18 "Category" (type=UNICODE_STRING*, size=0x4)
  arg 2: 0x2 (type=int, size=0x4)
  arg 3: 0x02c5f094 (type=<struct>*, size=0x4)
  arg 4: 0x90 (type=unsigned int, size=0x4)
  arg 5: 0x02c5f070 (type=unsigned int*, size=0x4)
succeeded =>
  arg 3: <NYI> (type=<struct>*, size=0x4)
  arg 5: 0x02c5f070 => 0x10 (type=unsigned int*, size=0x4)
retval: 0x0 (type=NTSTATUS, size=0x4)
```

Fig. 4.1. DrsTrace generated system call sample

Every system call was assigned with the unique number – a sequence of these numbers represents a system calls sequence produced by the specific malware or benign sample. A special tool was developed by the authors to extract the required number of system calls from the SQLite database.

System calls by benign applications were added to two sets:

1. Set of six classes (five malicious and one benign) – to be used in the classification accuracy test, i.e., assigning the activity to legal or to one of the five classes of malware programs;
2. Set of two classes (malware and benign) – to be used in the anomaly detection test, i.e., determining if the activity is malicious or not.

For both of these sets, additional subsets were generated, in which sequences of repeated system calls longer than 3 were replaced with a maximum of 2 repeated system calls (e.g. sequence “4655532” was transformed into “465532”) according to the recommendation in (Kolosnjaji et al., 2016).

Consequently, 66 sets (files) in total for training and testing were generated. The labelling of sets is presented in Table 4.1. It is necessary to mention that sets with removed repeated sequences had fewer samples. The main reason is that almost all system calls were identical (e.g. one of the malware samples contained only calls to NtCreateFile).

Table 4.1. Training and testing sets labelling

Set label	Sequences variations	Comments
AllMalware	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Only malware samples
AllMalware2	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Only malware samples with no more than two identical sequences in repetition
AllMalwarePlus-Clean	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Malware samples plus and benign samples as additional class
AllMalwarePlus-Clean2	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Malware samples and benign samples as an additional class with no more than two identical sequences in repetition
MalwarePlusClean	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Only two classes to train: Malware and Benign
MalwarePlus-Clean2	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Only two classes to train: Malware and Benign samples with no more than two identical sequences in repetition

Datasets AllMalware and AllMalware2 were selected to test if there is any difference in the accuracy as compared to simple ML methods performed earlier in Chapter 3. Commercially applicable accuracy of 92.4% was achieved with the Support Vector Machines method and comparable accuracy of 92.1%. was achieved by the decision tree method.

In this research, deep-learning methods were applied to check if they can provide higher accuracy using the same datasets.

Five malware families were selected from AWSCTD dataset for our experiment, each family with at least 100 samples of unique family representatives. A

family descriptor provided by Kaspersky was used. Table 4.2 shows the number of unique samples in each training set (family names according to Kaspersky).

Table 4.2. Data samples count by class

Class label	Samples Count
Trojan	1755
AdWare	4333
WebToolbar	618
Donwloader	710
DangerousObject ²	105
Benign	2350
Total	9871

SQLite database-based data was converted into easily readable CSV files. The sample data of 10 system calls long sequences file is presented in Table 4.3.

Table 4.3. CSV file data sample with all possible classes (SC – System Call)

1	2	3	4	5	6	7	8	9	10	SC Label
7	10	10	9	3	3	3	9	10	10	AdWare
20	20	20	10	10	9	9	9	18	11	Trojan
39	39	9	9	44	45	2	15	32	32	WebToolbar
10	10	48	9	9	36	11	11	11	11	Downloader
9	18	21	22	9	9	26	9	18	23	DangerousObject
10	40	26	26	29	29	13	9	16	41	Benign

The first ten numbers represent an unique system call number followed by the label for the malware family name or legal program (“Benign” label) if it is a benign sample.

To evaluate the efficiency of all models the following metrics were defined: accuracy, confusion matrix (CM), precision, recall, F-score, FPR, False Negative Rate (FNR), classification error, and training and testing times. The accuracy metric is calculated for both benign and malware data classes, while the precision, recall, F-score, Matthews Correlation Coefficient (MCC), FPR, FNR, and Error rate are calculated only for the malware class and represent the model capability

² DangerousObject – Malicious software that was detected by KL Cloud Technologies but was not classified exactly

to perform the intrusion/malware detection task. The criteria behind metrics selection is very simple – those metrics are recognised by the DL research community as the most value generating items.

4.3. Experimental Testbed Configuration

The source code was written using the Python language. The special Python libraries were used for the following tasks:

- For DL model creation and configuration – Keras 2.2.4 (Chollet & others, 2015) and Tensorflow 1.12.0 (Abadi et al., 2016) as the backend;
- For data loading, visualization, and pre-processing – Numpy 1.16.5, Pandas 0.24.2, Scikit 0.20.4, and Matplotlib 2.2.4.

All source code used in this paper is provided on the same repository as AWSCTD (Dainius Čeponis & Goranin, 2020a).

A relatively simple hardware configuration was used to support the above-mentioned software:

- CPU: Intel(R) Core (TM) i5-3570 3.80 GHz (4 Cores, 4 Threads);
- GPU: GTX 1070 (1920 Compute Unified Device Architecture (CUDA®) Cores);
- RAM: 16GB (DDR3);
- OS: Ubuntu 18.04.3 LTS.

4.4. Evaluation of Vanilla DL Models

Vanilla DL models will be used in the first part of experiments to identify the most accurate model. This section will cover the results of the following tests: malware classification task, family classification task, intrusion detection task and conclusions.

4.4.1. Model Selection Criteria and Configuration Parameters

The RNN and CNN methods were used in the experiment as they were determined as the most accurate and recognised by the research community in the section 1.3.4 of this thesis. Furthermore, they tend to generate better FPR results than most popular ML methods (NN, SVM or HMM) (Al Jallad et al., 2020; Bouzar-Benlabiod et al., 2020; Meng et al., 2017). The configuration for the ten system calls can be seen in Figure 4.2. RNN configuration with LSTM and GRU had three layers: Input, CuDNNLSTM or CuDNNGRU, and Dense. CNN had Input,

Convolution1D (with sliding window value of 6), GlobalMaxPooling1D, and Dense layers. The SVM model was also used to compare the results with previous results achieved in Chapter 3.

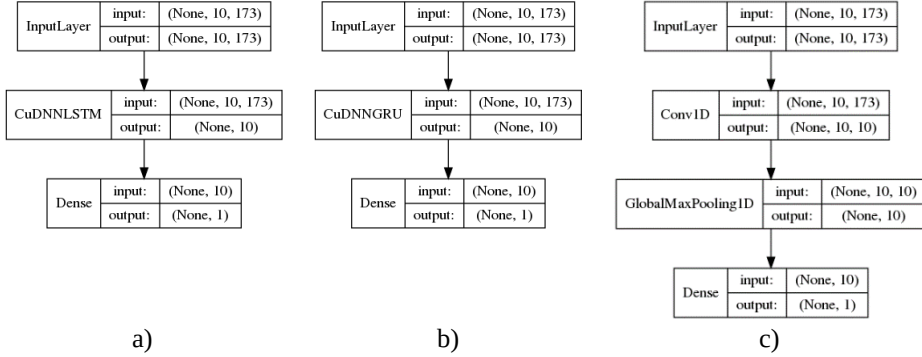


Fig. 4.2. The configuration of training LSTM, GRU, and CNN models for MalwarePlusClean data sample: a) LSTM; b) GRU; c) CNN

The following specific hyperparameters of models were changed as others remain with default values determined by the Keras library (Chollet & others, 2015):

- The model CNN is using system calls count as a *kernel value*: if 1000 system calls sequence is used, then kernel value will be equal to 1000. Sliding window (size of kernels) - 6. Padding: *same*. Activation: *tanh*. Kernel initializer – *glorot_uniform*;
- The LSTM and GRU models also use dynamic *unfold value* dependent on the number of system calls (one-to-one).

The following configuration parameters were used for all models:

- Epochs – 200 (training is stopped if no improvements for six epochs is observed);
- Batch size – 100;
- Fold number – 5 *splits* with *shuffle* parameter enabled;
- Optimizer – *Adam* with default parameters (Kingma & Ba, 2015);
- Metric to be evaluated by the model during training and testing – *accuracy* for two class and *categorical_accuracy* for multiclass;
- The activation function of the last layer (the layer of decision-making) – *sigmoid*. The reasoning behind this function – classification is executed in binary mode and for that type of classification sigmoid function produces better results than Softmax.

Despite relatively straightforward configuration, the models achieved over 90% classification accuracy with almost all data samples. An experiment with more complicated versions of the proposed models showed no valuable impact on the results. Deeper and more internal nodes containing models returned similar accuracy results in the margin of error and required longer training and classification times. For this reason, vanilla models were selected as they tend to generate usable results with less computation power and time consumption. The classifiers were trained and tested using a 5-fold cross-validation technique. Cross-validation is an overfitting preventing technique for evaluating predictive models by partitioning the original sample into a training set to train the model and a test set to evaluate it. The callback of EarlyStopping was used to stop the training process when it did not improve for six epochs (also prevents from overfitting). Furthermore, we used one-hot encoding to provide data for the training models. Ten system calls samples had unique 173 system calls (see Figure 4.2 for the value of input shape dimensions). Larger data samples had more unique system calls: for example, 400 had unique 488 system calls. In comparison, (Kolosnjaji et al., 2016) had only 60 unique system calls, which means that our dataset is more diverse.

4.4.2. Evaluation of Malware Classification Task

As stated earlier, the results achieved with DL methods were compared with the results achieved through classical ML methods in the Chapter 3. The data labelled AllMalware were used in that test. Although the original results of Chapter 3 have shown that SVM method can achieve 92.4% accuracy with the 100 first malicious system calls sequences, it can be seen that DL methods demonstrate significantly better results with the same dataset (see Table A.1 and Figure 4.3). The accuracy is calculated as follows: the method of machine learning is trained with a portion of the dataset (80%), whilst another portion of the dataset is used for testing with the trained model, i.e. data used for testing had not been used for training. Therefore, the percentage of correctly classified records is defined as the accuracy.

CNN achieved 92.8% (MAE of 0.05) accuracy on the same length of 100 sequence calls. It has also shown better accuracy for 200, 400, 600, 800, and 1000 system calls sequences than SVM (92.7% vs 89.6%, 93.0% vs 87.3%, 93.1% vs 86.1%, 93.0% vs 84.7%, and 93.1% vs 83.2% respectively). This implies that a practically applicable accuracy (>90%) can be maintained even with larger datasets by applying CNN.

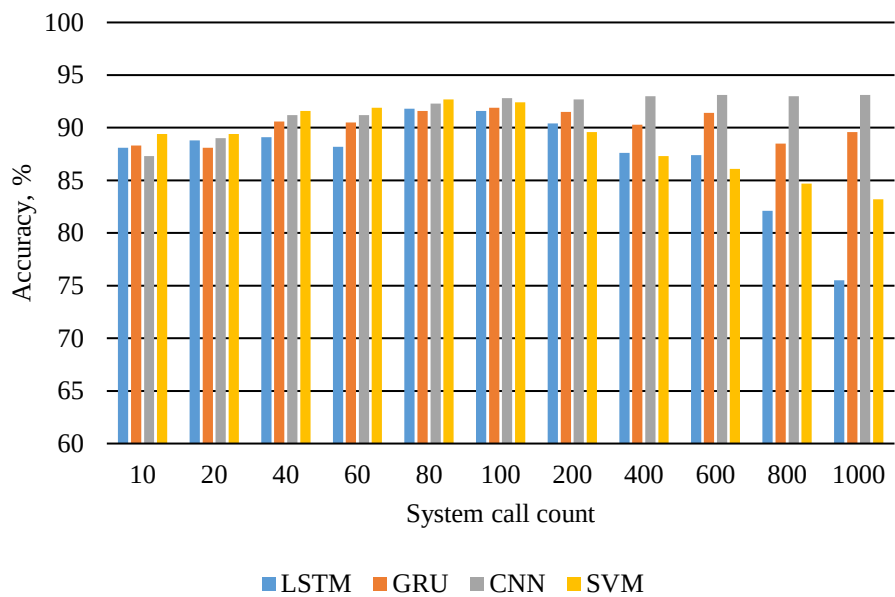


Fig. 4.3. Classification accuracy (AllMalware set)

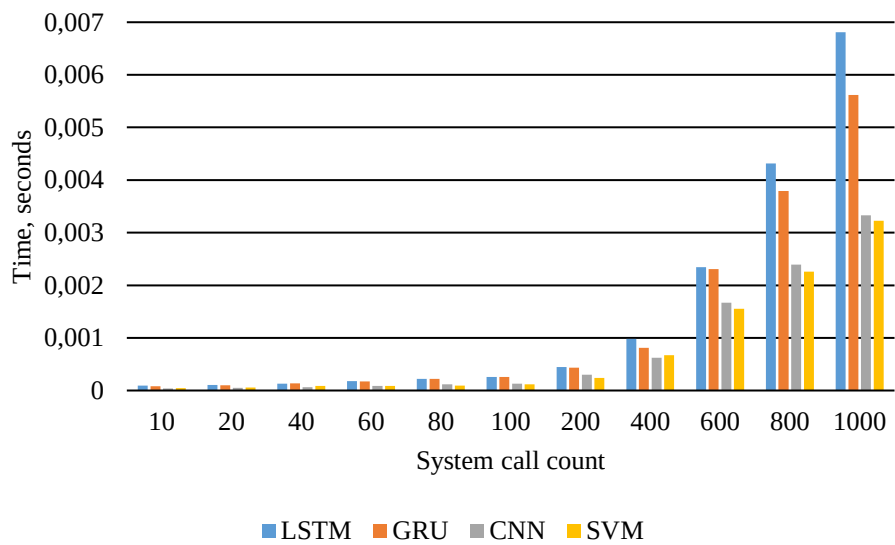


Fig. 4.4. Classification time (AllMalware set)

Sequence-based DL methods (LSTM and GRU) demonstrated worse results than SVM – the achieved accuracy was equal to 88.1% (MAE of 0.09) and 88.3% (MAE of 0.08) on the first ten system calls, respectively. CNN not only demonstrated better accuracy but also achieved a somewhat similar classification time compared to the much simpler SVM model. Similar times were maintained even with a larger data sample. In terms of accuracy, SVM demonstrates degrading results in comparison with CNNs.

The data presented in Table A.1 and Figure 4.4 shows that classification time starts to grow exponentially when longer than foremost 100 system calls are used. The RNN-based models required longer times in comparison with CNN-based and SVM-based models. For example, 1000 system calls classification with CNN model was 48% faster than with LSTM model.

A classification performance, FPR, PNR and error rate metrics for foremost 1000 system calls classification with AllMalware set have been calculated and are presented in Table 4.4.

Table 4.4. Classification performance metrics for 1000 system calls (AllMalware set)

Metric	LSTM	GRU	CNN	SVM
Precision	0.663	0.678	0.739	0.707
Recall	0.618	0.628	0.736	0.591
F-1	0.627	0.645	0.737	0.630
FPR	0.056	0.049	0.024	0.073
FNR	0.382	0.372	0.264	0.409
Error rate	0.145	0.146	0.071	0.168

The results repeat a conclusion that a CNN based model is somewhat the best-performing model in comparison with other DL and ML models as it produces the highest precision (0.739 when the average is 0.697), recall (0.736 when the average is 0.643) and F-1 (0.737 when the average is 0.660) scores. Also, it shows the lowest FPR (2.4% when the average is 5%), FNR (26.4% when the average is 35.7%) and error rate (0.071 when the average is 0.133) values in comparison with other DL and ML models, bearing in mind that MEA of CNN for 1000 system calls was 0.03. All other compared models produced higher or equal MAE values: LSTM – 0.10; GRU – 0.08; SVM – 0.44. There is a 95% likelihood that the average confidence interval [0.053, 0.095] covers the true classification error of the best performing CNN-based model on unseen data.

4.4.3. Evaluation of the Family Classification Task

Benign samples were introduced next in the training process. As described earlier, two sets were used in tests: with repetitive system calls (AllMalwarePlusClean) and without repetitive calls (AllMalwarePlusClean2). The introduction of a new family to the training set resulted in a decrease in the accuracy (see Table A.2, Figure 4.5 and Figure 4.6).

The removal of repetitive system calls increased the accuracy of the results. The best accuracy of 93.9% (MAE of 0.03) was achieved with CNN and 1000 of the first system calls (AllMalwarePlusClean2 data sample). However, a relatively similar outcome of 93.5% (MAE of 0.03) was obtained with only 600 of systems calls, which required much less time for training. As results for 600 and 1000 system calls differ only in the error rate, it can be said that a set of 600 system calls is more preferable for practical applications. On smaller sets, the results by CNN were low (86.9% with MAE of 0.06) but still higher than the results by LSTN and GRU (85.8% and 85.6% and MAE of 0.07 respectively).

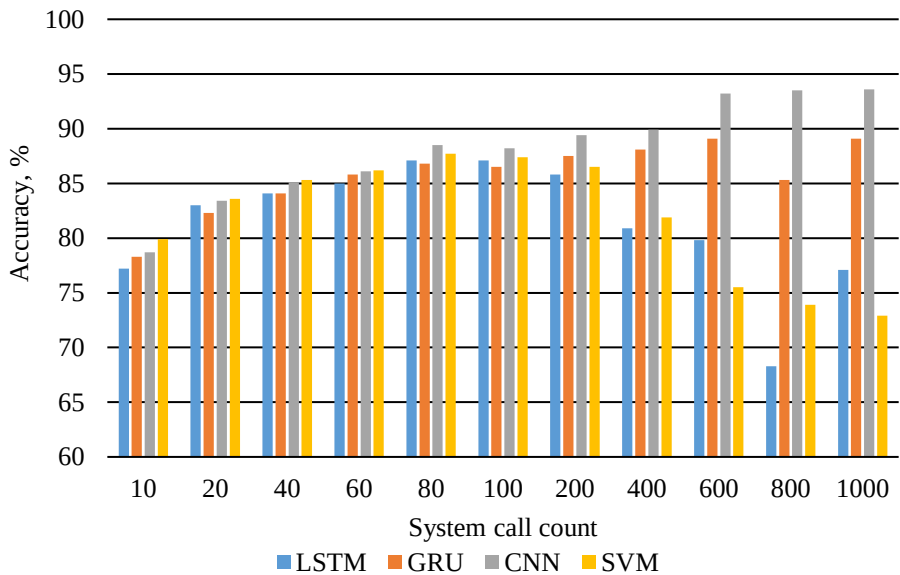


Fig. 4.5. Classification accuracy (AllMalwarePlusClean set)

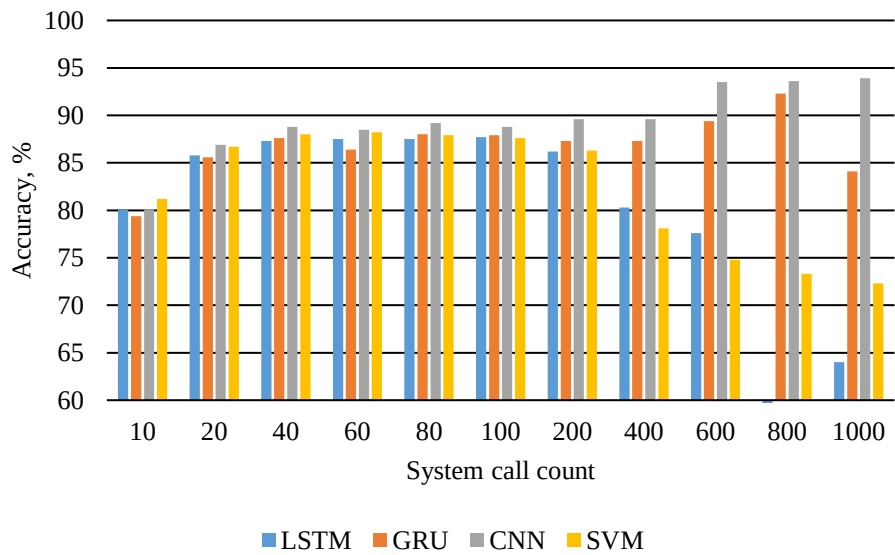


Fig. 4.6. Classification accuracy (AllMalwarePlusClean2 set)

A classification performance, FPR, PNR and error rate metrics for foremost 1000 system calls classification with AllMalwarePlusClean2 set have been calculated and are presented in Table 4.5.

Table 4.5. Classification performance metrics for 1000 system calls (AllMalwarePlusClean2 set)

Metric	LSTM	GRU	CNN	SVM
Precision	0.632	0.742	0.778	0.728
Recall	0.462	0.743	0.779	0.540
F-1	0.640	0.738	0.778	0.581
FPR	0.088	0.023	0.015	0.085
FNR	0.538	0.257	0.221	0.460
Error rate	0.310	0.099	0.062	0.277

The results repeat a conclusion that CNN based model is somewhat best-performing model in comparison with other DL and ML models as it produces the highest precision (0.778 when the average is 0.720), recall (0.779 when the average is 0.631) and F-1 (0.778 when the average is 0.684) scores. Also, it shows the lowest FPR (1.5% when the average is 5.3%), FNR (22.1% when the average is

36.9%) and error rate (0.062 when the average is 0.187) values in comparison with other DL and ML models in mind that MEA of CNN for 1000 system calls was 0.03. All other compared models produced higher or equal MAE values: LSTM – 0.15; GRU – 0.05; SVM – 0.62. The additional (Benign) class helped to boost CNN-based model performance but lowered results for other models because average results are worse than with AllMalware set. There is a 95% likelihood that the average confidence interval [0.047, 0.078] covers the true classification error of the best performing CNN-based model on unseen data.

Figure 4.7 presents the family classification task results by family in case of a set of 100 system calls.

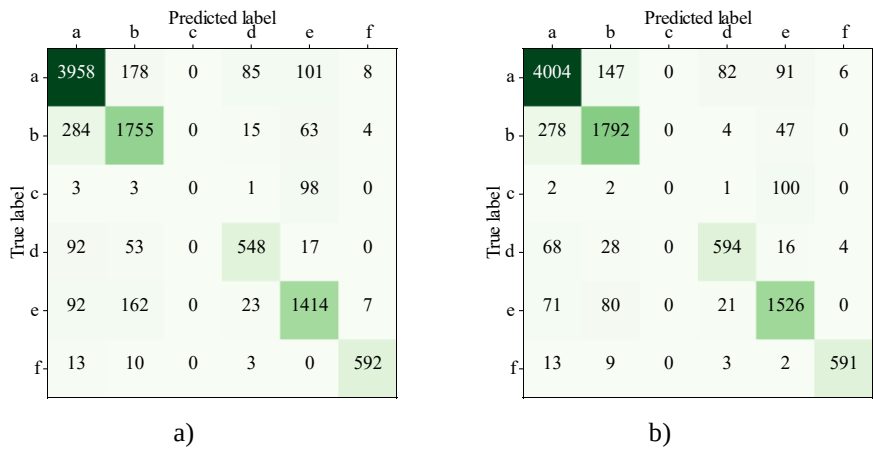


Fig. 4.7. Confusion matrix of the GRU and CNN methods for the 100 system calls sequence of the AllMalwarePlusClean data sample: a) GRU; b) CNN. Class labels: AdWare (a); Benign (b); DangerousObject (c); Downloader (d); Trojan (e); WebToolbar (f)

It can be clearly seen that the number of samples in the training data has a huge impact on the correct classification. WebToolbar, Downloader, and DangerousObject labelled samples have more incorrect labels assignments than AdWare, Benign and Trojan. The lowest classification score has a DangerousObject class – zero. That outcome was expected, since Kaspersky itself is not sure about the label, and in prior Chapter 3, even the best performing SVM model also generated zero correct classification results for this class. Both models of GRU and CNN classified this family as belonging to the Trojan class. Even CNN model, which generates the best performance (90.0% for that specific data collection), shows that DangerousObject class should be labelled as Trojan.

4.4.4. Evaluation of the Intrusion Detection Task

Finally, the intrusion detection task was performed, i.e. the applicability of DL methods for determining if an activity is malicious or benign was evaluated (see Table 4.8). All malicious system calls were merged into one family, and the second family contained only benign system calls. As in the previous case, sets both with repetitive system calls (MalwarePlusClean) and with removed repetitive system calls (MalwarePlusClean2) were used.

4.4.4.1. Accuracy

In this case, the set without repetitive system calls produced comparable results with the full set. This implies that the system calls minimization technique is effective and can be used to achieve better accuracy in family classification and intrusion detection tasks whilst minimizing the model training time.

Accuracies of 94.5% (MAE of 0.07), 94.8% (MAE of 0.06) and 99.3% (MAE of 0.01) were obtained by CNN for the 100, 400 and 1000 first system calls respectively (MalwarePlusClean2) (Table A.3, Figure 4.8 and Figure 4.9). CNN has also shown the best results for all data samples in the two-class classification task (i.e. intrusion detection) of all ML methods used: usable accuracy of 93.2% (MAE of 0.09) was obtained even for the 20 first system calls.

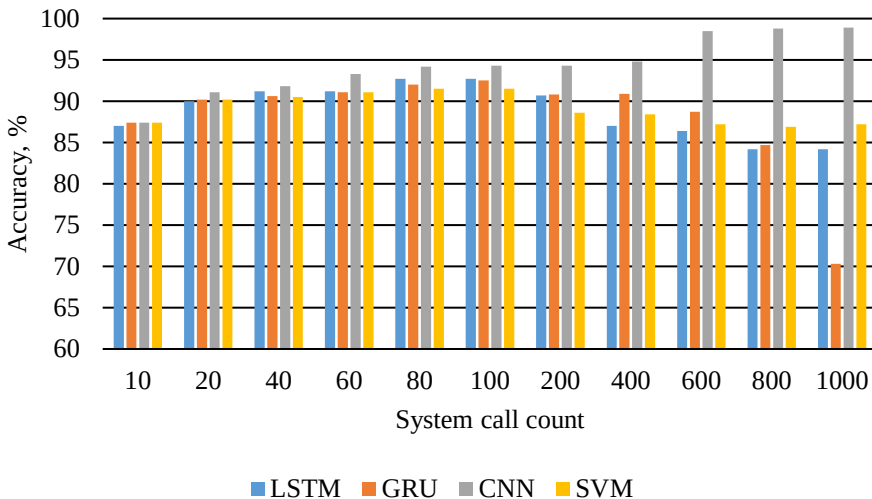


Fig. 4.8. Classification accuracy (MalwarePlusClean set)

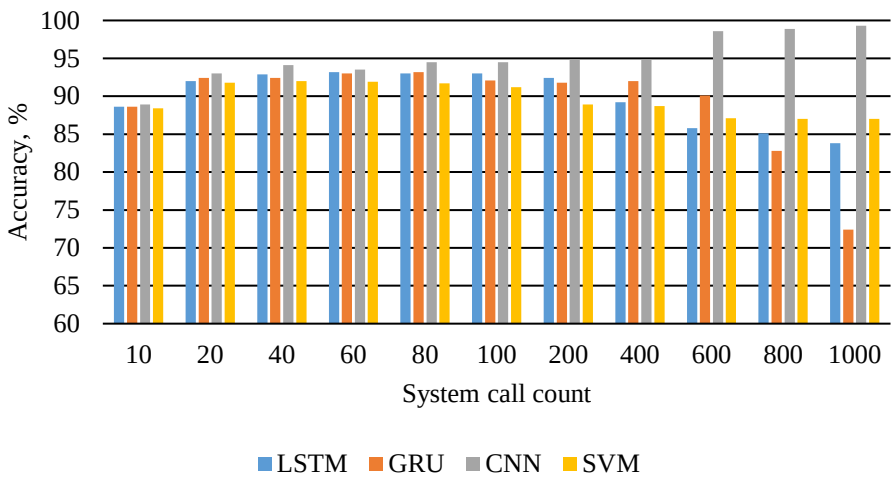


Fig. 4.9. Classification accuracy (MalwarePlusClean2 set)

4.4.4.2. Comparison of the performance metrics

Table 4.6. Classification performance metrics for 1000 system calls (MalwarePlusClean2 set)

Metric	LSTM	GRU	CNN	SVM
Precision	0.718	0.591	0.987	0.931
Recall	0.671	0.614	0.986	0.648
F-1	0.689	0.597	0.987	0.691
FPR	0.329	0.386	0.014	0.352
FNR	0.121	0.244	0.006	0.065
Error rate	0.168	0.276	0.008	0.130

A classification performance, FPR, PNR and error rate metrics for foremost 1000 system calls classification with MalwarePlusClean2 set have been calculated and are presented in Table 4.6. The results repeat a conclusion that CNN based model is somewhat best-performing model (as concluded with other sets) in comparison with other DL and ML models as it produces the highest precision (0.987 when the average is 0.807), recall (0.986 when the average is 0.730) and F-1 (0.987 when the average is 0.741) scores. Also, it shows the lowest FPR (1.4% when the average is 27.0%), FNR (0.06% when the average is 10.9%) and error rate (0.008 when the average is 0.146) values in comparison with other DL and

ML models bearing in mind that MEA of CNN for 1000 system calls was 0.03. All other models compared produced higher or equal MAE values: LSTM – 0.26; GRU – 0.19; SVM – 0.19. The conversion of malware family classes to one resulted in higher performance results for all models. Only FPR values decreased drastically for RNN-based and SVM-based models when compared against AllMalwarePlusClean2 dataset. There is a 95% likelihood that the average confidence interval [0.002, 0.015] covers the true classification error of the best performing CNN-based model on unseen data.

4.4.4.3. Comparison of the CMs

In the two-class confusion matrices (see Figure 4.10), it can be seen that fewer Malware samples are assigned to Benign by CNN as compared to GRU results.

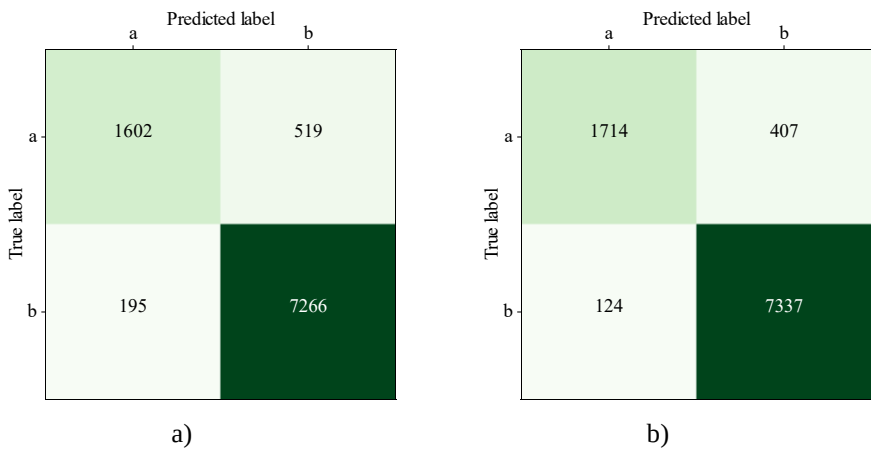


Fig. 4.10. Confusion matrix of the GRU and CNN methods for the 100 system calls sequence of the MalwarePlusClean data sample: a) GRU; b) CNN. Class labels: Benign (a); Malware (b)

This characteristic is essential in the target field; malignant actions classification as benign must be minimal for the IDS. Benign samples decision is somewhat comparable for the GRU and CNN models.

4.4.4.4. Comparison of the ROC and AUC

GRU and CNN models demonstrate outstanding results in the means of the ROC and AUC. In Figure 4.11 and Figure 4.12, the ROC diagrams with the combination of the AUC values are represented for the MalwarePlusClean samples with

the 100 system calls. ROC and AUC are displayed for every fold. Mean ROC and AUC are represented with the blue line.

The best mean AUC value of 0.98 is generated by the CNN model for both classes, i.e. there is a 98% chance that the model will be able to distinguish between Malware class and Benign class. The comparable result of 0.97 is achieved by GRU. A high AUC value indicates that both models (GRU and CNN) have good class separation capacity.

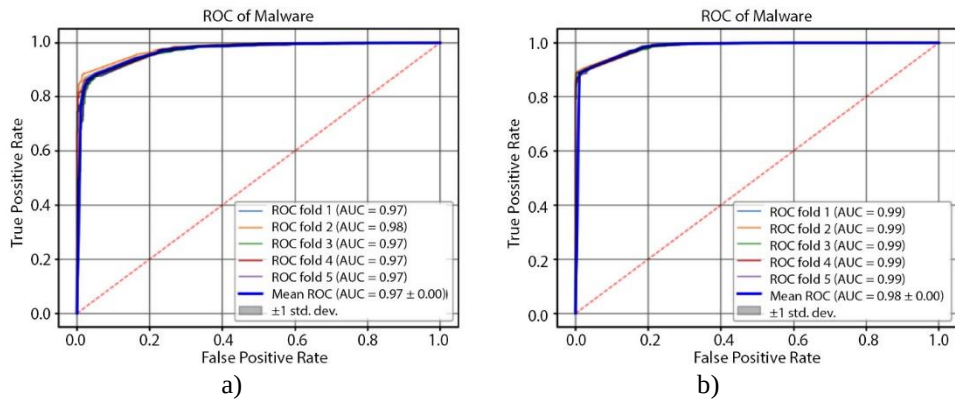


Fig. 4.11. ROC diagrams of the Malware class for the 100 system calls of MalwarePlusClean data: a) GRU; b) CNN

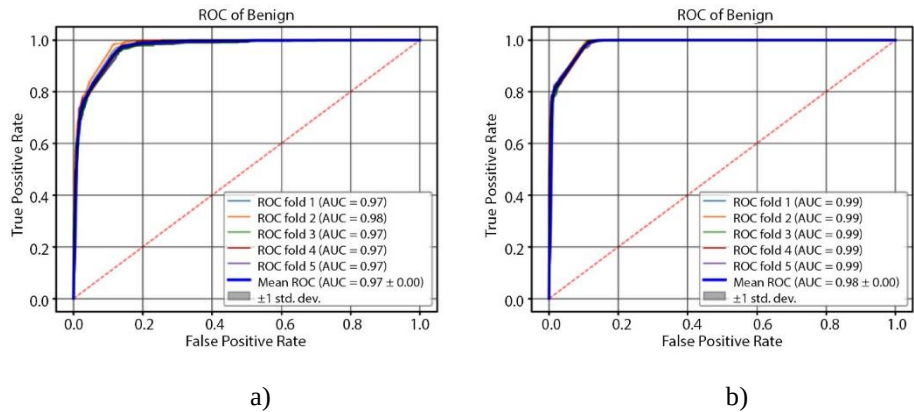


Fig. 4.12. ROC diagrams of the Benign class for the 100 system calls of MalwarePlusClean data: a) GRU; b) CNN

4.4.4.5. Evaluation of Model Configuration on Training

The evaluation of system call sequence size on the model training time was performed on the AllMalwarePlusClean set. Figure 4.13 presents the training time for LSTN, GRU, and CNN with sequences of 10, 100, 200, and 400 system calls. It can be seen that the increase of sequence length results in an exponential increase of training time, making extremely long sequences not applicable for everyday use.

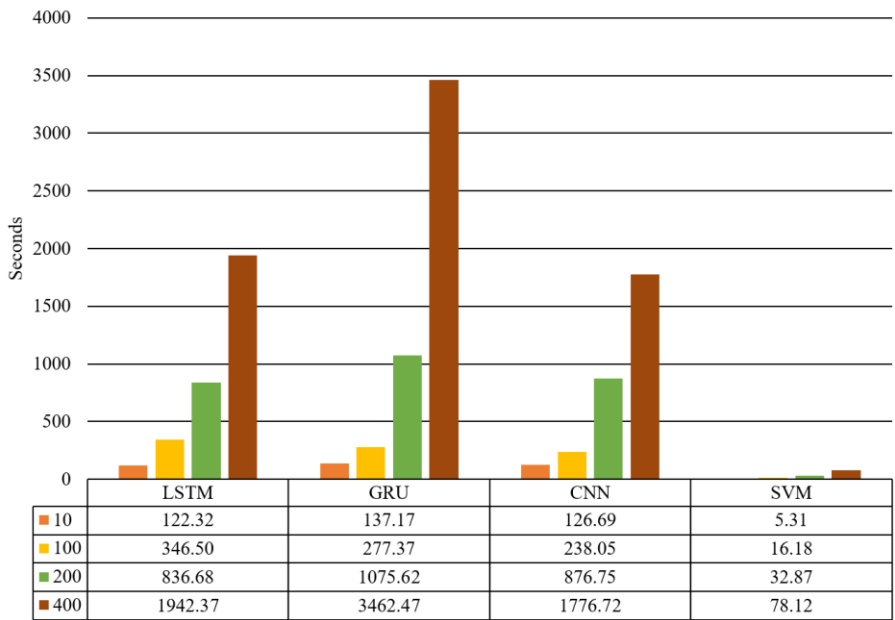


Fig. 4.13. Training time comparison of the AllMalwarePlusClean data collection. 10, 100, 200 and 400 sequence calls as data points

GRU training time was equal to 57.7 minutes with the sequence of 400 system calls. The best performing CNN model training took 29.6 minutes with the same dataset. In comparison, 100 system calls sequences training time is much faster. For GRU and CNN, it took 4.6 and 3.9 minutes, respectively.

The evaluation of data model size impact on training time leads to the conclusion that using the first 100 system calls sequences is an optimal solution in terms of time and accuracy balance.

4.4.4.6. The Comparison of the Number of Epochs Needed to Reach the Saturation

The classification accuracy vs. number of epochs needed to reach the saturation measurement is presented in Figure 4.14.

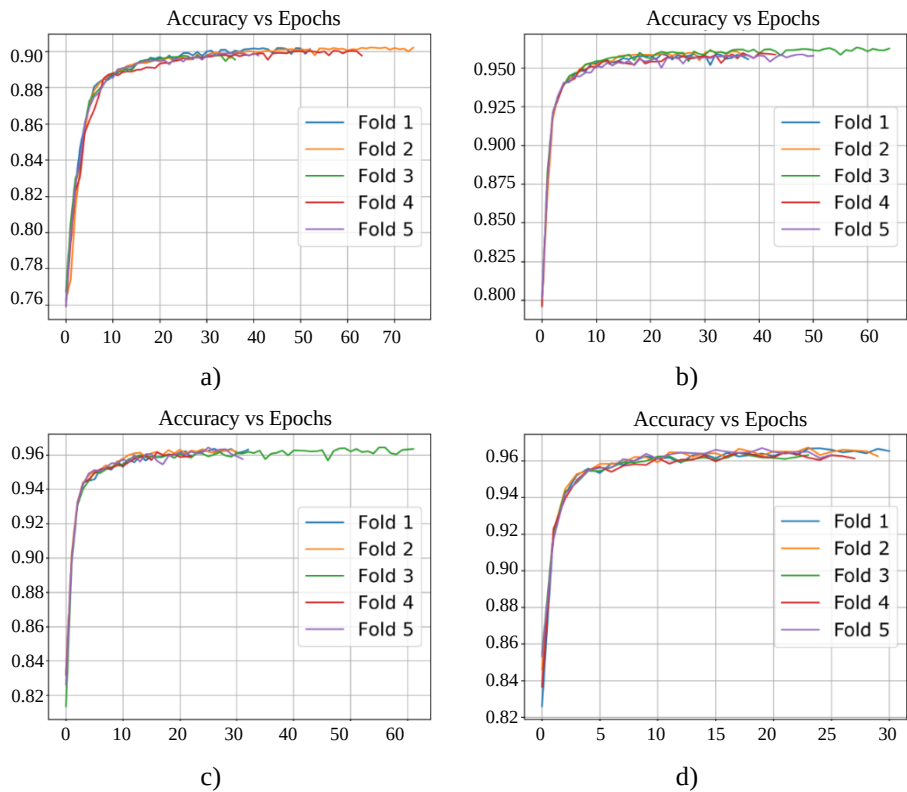


Fig. 4.14. Training time comparison of the AllMalwarePlusClean data collection. 10, 100, 200 and 400 sequence calls as data points: a) CNN 10; b) CNN 100; c) CNN 200; d) CNN 400

As it can be seen, there is a reverse dependency of the number of epochs before saturation on the system call sequence length, e.g. for the top-performing CNN model, 75 epochs are required to train 10 system calls, and only 30 epochs are required for 400 system calls.

4.4.4.7. Evaluation of Equal Error Rate

The computed equal error rate (EER) values for the DL methods (LSTM, GRU, and CNN) can be seen in the Table 4.7. When comparing models, lower EER means better performance of the model. In our case, CNN shows best values of 9.7% and 4.8% for the Benign and Malware classes, respectively.

Table 4.7. EER values in percent of the DL methods generated for the MalwarePlusClean of 100 system calls dataset

DL Method	Benign	Malware
LSTM	10.9	7.5
GRU	11.0	8.0
CNN	9.7	4.8

4.5. Evaluation of Advanced DL Models

In this section, the experimental results for the metrics (accuracy, CM, precision, recall, F-score, FPR, FNR, classification error, and training and testing times), defined in the previous chapter, are presented, as well as evaluation of the significance of achieved results by dual-flow models in comparison with vanilla single-flow models. Additionally, the number of epochs needed to reach the saturation and CMs are provided and discussed.

4.5.1. Advanced DL Models Used and Configuration Parameters

In total, seven DL models were used in the intrusion detection experiments. State-of-the-art dual-flow LSTM-FCN (Figure 4.19) and GRU-FCN (Figure 4.20) models were selected since typically more complex DL configurations tend to generate better results (Szegedy et al., 2015). Those models consist of two flows that are connected to the last decision-making layer: the CNN and RNN flow (LSTM or GRU, respectively). They have been designed for use in time-series datasets, which often are univariate (contains only one observed parameter). System calls from AWSCTD fit perfectly for such methods.

Apart from that, several additional experimental models, i.e., not described elsewhere previously, were tested. One of the main criteria influencing the model configuration selection was the least possible number of internal nodes in order to ensure reasonable training/testing times while preserving high accuracy and other metrics values. The selection of models was also based on results achieved by

Reference (Deng & Platt, 2014; Wen et al., 2017; Yuanyuan & Zhibin, 2018), in which DL models (CNN and RNN) were combined in a linear ensemble manner. Combined models returned better accuracy results than separate usage of CNN and RNN models on speech, image, and infrared spectroscopy analysis. The aforementioned research encouraged us to use the linear ensemble with our previously tested Chapter 4 models and already recognized as state-of-the-art (Zhiguang Wang et al., 2017) models:

- Single-flow model (Figure 4.18), which contains only CNN flow extracted from LSTM-FCN and was originally named FCN in Reference (Zhiguang Wang et al., 2017).
- Single-flow CNN model (Figure 4.15), which demonstrated the best results in our previous experiments in first part of Chapter 4 and is named AWSCTD-CNN- dynamic (D) (dynamic value of kernels parameter). The components of the model are marked in orange colour to better understand the combination of later models.
- New single-flow CNN model (Figure 4.15), named AWSCTD-CNN-S (static value of kernels parameter).
- New dual-flow model (Figure 4.16), named AWSCTD-CNN-LSTM, combined of AWSCTD-CNN-D and LSTM. The orange colour represents AWSCTD-CNN-D flow, and the green colour represents LSTM flow.
- New dual-flow model (Figure 4.17), named AWSCTD-CNN-GRU, combined of AWSCTD-CNN-D (orange colour) and GRU (green colour).

It can be seen that the LSTM-FCN, GRU-FCN, and FCN consist of a more complex structure in comparison with the AWSCTD-CNN, AWSCTD-CNN-LSTM, and AWSCTD-CNN-GRU models. This feature implies that those models can provide better results in intrusion detection, as suggested in Reference (Elsayed et al., 2019; Karim et al., 2017).

The models were grouped into two families by origin:

- FCN family:
 1. LSTM-FCN;
 2. GRU-FCN;
 3. FCN.
- AWSCTD family:
 1. AWSCTD-CNN-D;
 2. AWSCTD-CNN-LSTM;
 3. AWSCTD-CNN-GRU;
 4. AWSCTD-CNN-S.

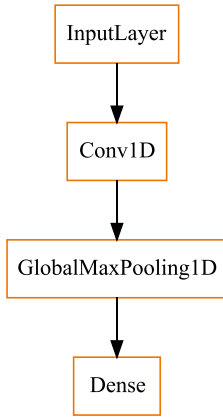


Fig. 4.15. Diagram of AWSCTD-CNN-dynamic (D) and AWSCTD-CNN-static (S) models

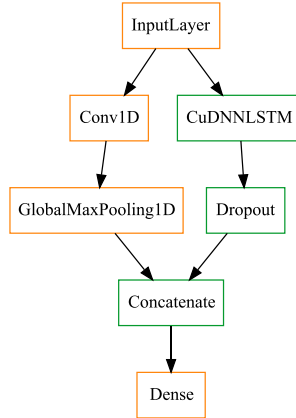


Fig. 4.16. Diagram of attack-caused Windows OS system calls traces (AWSCTD)-convolutional neural network (CNN)-LSTM model

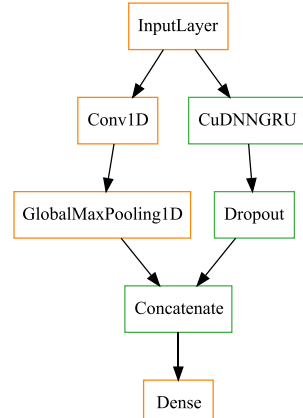


Fig. 4.17. Diagram of AWSCTD-CNN-GRU model

The following original specific parameters were used for FCN family models:

- The models from FCN family contain original configuration as described in original papers (Elsayed et al., 2019; Karim et al., 2017);
- The FCN-GRU part has 8 unfolds and uses an aggressive dropout coefficient of 0.8 (to overcome overfitting);
- The CNN part in those models has the following parameters:
 - Kernels – 128, 256, 128;
 - Kernel sizes – 8, 5, 3;
 - Kernel initializer – *he_uniform*;
 - Activation – *tanh*;
 - Batch normalization parameter of epsilon is 0.01.

The following specific parameters were used for AWSCTD family models:

- The dynamic model AWSCTD-CNN-D is using system calls count as a kernel value: if 1000 system calls sequence is used, then kernel value will be equal to 1000;
- The AWSCTD-CNN-LSTM and AWSCTD-CNN-GRU models also use dynamic unfold value dependent on the number of system calls (one-to-one);
- The dropout value is identical to the FCN family – 0.8;

- The static model AWSCTD-CNN-S is using 256 as kernels number for all system calls sequences. The value of 6 is used as the size of kernels since it has been proved as optimal by other authors (Tandon Gaurav & Philip K. Chan, 2005). Kernel initializer – *glorot_uniform*. Activation – *tanh*.

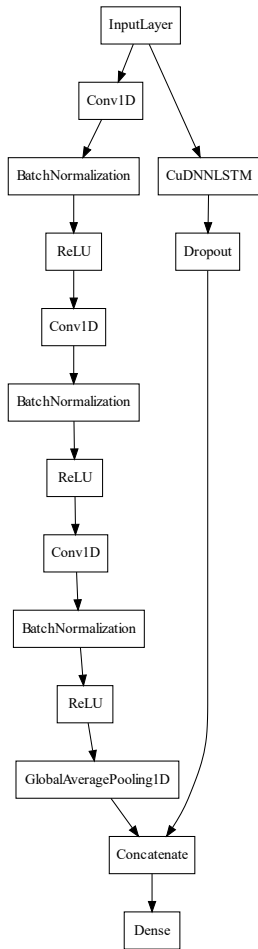


Fig. 4.18. Diagram of long short-term memory fully convolutional network (LSTM-FCN) model. CuDNN = CUDA® (Compute Unified Device Architecture) Deep Neural Network library (cuDNN); ReLU = rectified linear unit

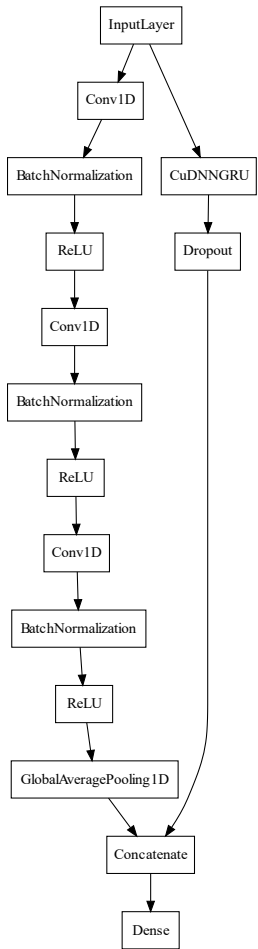


Fig. 4.19. Diagram of gated recurrent unit (GRU)-FCN model

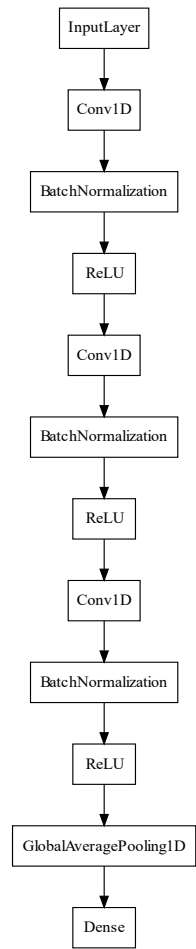


Fig. 4.20. Diagram of FCN model

The following configuration parameters were used for all models:

- Epochs – 200 (training is stopped if no improvements for six epochs is observed);
- Batch size. This value has been selected heuristically to support bigger ML models due to the small amount of RAM available. It is worth mentioning that previous research has used a value of 100. The value of batch is crucial for CNN accuracy results – the higher value is producing better results but is consuming more memory and requires more computational power (Radiuk, 2018);
- Fold number – 5 splits with shuffle parameter enabled;
- Objective function (or loss function) – binary_crossentropy;
- Optimizer – Adam with default parameters (Kingma & Ba, 2015);
- Metric to be evaluated by the model during training and testing – accuracy;
- The activation function of the last layer (the layer of decision-making) – sigmoid. The reasoning behind this function – classification is executed in binary mode and for that type of classification sigmoid function produces better results than Softmax.

4.5.2. Evaluation of the Intrusion Detection Task

In this section, the intrusion detection task was performed, i.e. the applicability of advanced DL methods for determining if an activity is malicious or benign was evaluated. The reasoning behind this task selection is the following: the primary job of the IDS is to detect intrusion as the event and notify security personnel. Later, an event can be prioritized by the type of detected malicious application. All malicious system calls were merged into one family, and the second family contained only benign system calls. A set with removed repetitive system calls (MalwarePlusClean2) was used as it produced the most promising results in previous tasks. The intrusion detection task has been selected as the main target of the dissertation is intrusion itself.

4.5.2.1. Accuracy

The accuracy metric demonstrates somewhat similar tendencies for all models - longer system calls sequences tend to provide better results (Table A.4).

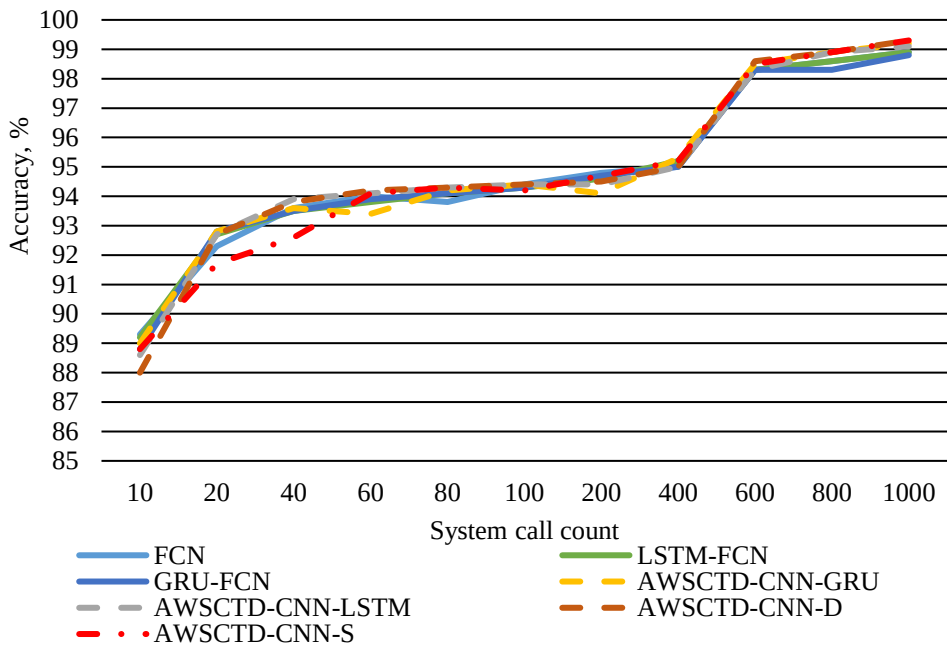


Fig. 4.21. Classification accuracy

None of the models reached the 90% accuracy for the sequence of 10 system calls, and the 90% threshold is achieved with 20 system calls sequence by all models. The sequence of 600 system calls can be considered as a threshold that starts providing practically applicable accuracy of more than 98% for all models (Figure 4.21).

All models are performing almost the same, and there is no clear winner (results vary from 98.8% to 99.3% (MAE varies from 0.01 to 0.02) for the sequence of 1000 system calls). None of the dual-flow FCN family models demonstrates significantly better results than single-flow models, except cases with short system call sequences (up to 10 system calls). The addition of an RNN (LSTM or GRU) layer to the single-flow models also does not result in the expected better accuracy. For longer sequences, single-flow models are demonstrating better accuracy. The reduction of batch size from 100 to 20 had no statistically significant impact on the accuracy of the AWSCTD-CNN-D model.

4.5.2.2. Precision, Recall, and F-score

The values of the precision metric demonstrate (Table 4.8) that all models are detecting malware or intrusion with a probability varying from 0.888 for 10 system calls up to 0.996 for 1000 system calls, i.e., correctness of models can be considered as very high.

Table 4.8. Precision metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.897	0.906	0.896	0.888	0.891	0.890	0.900
20	0.946	0.932	0.931	0.932	0.939	0.931	0.933
40	0.955	0.941	0.942	0.943	0.937	0.938	0.939
60	0.943	0.946	0.945	0.952	0.944	0.942	0.940
80	0.945	0.945	0.947	0.946	0.944	0.945	0.944
100	0.947	0.948	0.945	0.946	0.947	0.947	0.945
200	0.950	0.950	0.955	0.958	0.952	0.948	0.951
400	0.954	0.959	0.955	0.955	0.952	0.953	0.953
600	0.990	0.990	0.989	0.992	0.992	0.989	0.990
800	0.992	0.993	0.990	0.992	0.994	0.991	0.989
1000	0.995	0.994	0.995	0.996	0.993	0.994	0.992

Table 4.9. Recall metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.965	0.941	0.964	0.982	0.980	0.981	0.961
20	0.946	0.977	0.978	0.977	0.962	0.976	0.977
40	0.949	0.982	0.981	0.977	0.983	0.981	0.980
60	0.984	0.982	0.981	0.965	0.981	0.980	0.984
80	0.985	0.984	0.983	0.982	0.979	0.983	0.983
100	0.981	0.983	0.986	0.985	0.983	0.983	0.984
200	0.986	0.983	0.975	0.969	0.984	0.986	0.984

End of the Table 4.9

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
400	0.988	0.981	0.984	0.988	0.988	0.990	0.987
600	0.991	0.992	0.990	0.990	0.987	0.989	0.989
800	0.994	0.993	0.996	0.994	0.990	0.991	0.989
1000	0.997	0.997	0.994	0.995	0.993	0.993	0.993

Table 4.10. F-score metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.930	0.923	0.928	0.932	0.934	0.933	0.929
20	0.946	0.954	0.954	0.954	0.950	0.953	0.954
40	0.952	0.961	0.961	0.960	0.960	0.959	0.959
60	0.963	0.964	0.963	0.958	0.962	0.961	0.962
80	0.964	0.964	0.965	0.964	0.961	0.963	0.963
100	0.964	0.965	0.965	0.965	0.965	0.965	0.964
200	0.967	0.966	0.965	0.963	0.968	0.967	0.967
400	0.971	0.970	0.969	0.971	0.970	0.971	0.969
600	0.991	0.991	0.990	0.991	0.989	0.989	0.989
800	0.993	0.993	0.993	0.993	0.992	0.991	0.989
1000	0.996	0.995	0.994	0.995	0.993	0.993	0.992

The recall metric is designed to take evaluate the classification accuracy of intrusions/malware types by classes. In our case, the recall values are higher than corresponding precision values (Table 4.9). This can be explained by the fact that, in the case of a precision metric, benign system call class has a smaller size and decrease the overall accuracy compared to the recall metric.

The F-score metric is calculated from the precision and recall as the harmonic mean of both values. The distribution of this metric is also similar to the accuracy (Table 4.10) – the longest system calls sequences are producing the best results.

The results of precision, recall, and F-score (Table 4.10) demonstrate that all of the tested models are performing almost the same, and advantage of more complex models cannot be observed. The sequence of 600 system calls remains as the threshold for obtaining noticeably better and practically applicable results.

4.5.2.3. MCC

The value of MCC is calculated from all CM parameters. This has an advantage for our imbalanced data. As it can be seen in Table 4.11, all MCC coefficients are close to 1. That means that models' predictions are matching with actual results.

Table 4.11. Matthews Correlation Coefficient (MCC) metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.664	0.648	0.659	0.672	0.680	0.676	0.666
20	0.766	0.788	0.786	0.789	0.777	0.785	0.789
40	0.788	0.816	0.818	0.812	0.810	0.806	0.808
60	0.821	0.826	0.820	0.804	0.817	0.811	0.814
80	0.824	0.825	0.826	0.822	0.811	0.821	0.819
100	0.822	0.828	0.828	0.828	0.828	0.826	0.824
200	0.831	0.826	0.822	0.816	0.833	0.827	0.831
400	0.840	0.836	0.833	0.843	0.834	0.842	0.833
600	0.950	0.954	0.944	0.952	0.943	0.943	0.944
800	0.963	0.963	0.962	0.963	0.955	0.954	0.942
1000	0.977	0.975	0.970	0.974	0.963	0.964	0.959

The 600 system calls sequence repeatedly shows its importance – it improves MCC value by 13% in comparison with 400 system calls sequence (in case of the relatively best performing AWSCTD-CNN-S model).

4.5.2.4. FPR and FNR

In the context of intrusion detection, the false/missed alarm (FPR or FNR) is an important metric: a high rate of false alarms disturbs the security staff attention and increases the chance of a missed attack. The minimization of such events is an essential field of IDS research (Nguyen et al., 2014). This metric demonstrates

(Table 4.12) that AWSCTD family models have an advantage against the FCN family: the AWSCTD-CNN-GRU model has the best FPR of 0.018 for 1000 system calls sequence, while LSTM-FCN has the best value of 0.027 for the same sequence, i.e., 50% worse.

Table 4.12. False positive rate (FPR) metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.368	0.324	0.373	0.414	0.397	0.403	0.357
20	0.179	0.238	0.243	0.237	0.208	0.240	0.236
40	0.156	0.214	0.209	0.205	0.227	0.224	0.220
60	0.213	0.199	0.203	0.176	0.209	0.215	0.225
80	0.211	0.208	0.203	0.206	0.212	0.210	0.213
100	0.200	0.198	0.209	0.205	0.201	0.203	0.211
200	0.205	0.202	0.178	0.167	0.196	0.211	0.199
400	0.198	0.177	0.194	0.194	0.209	0.203	0.204
600	0.041	0.041	0.048	0.035	0.035	0.047	0.044
800	0.033	0.031	0.042	0.035	0.028	0.038	0.046
1000	0.022	0.026	0.022	0.018	0.030	0.027	0.035

The static (AWSCTD-CNN-S) and dynamic (AWSCTD-CNN-D) models demonstrate the lowest FNR values for 1000 system calls – 0.003 (Table 4.13). In comparison, the best achieved FNR result by the FCN family model is equal to 0.007 and is almost 60% worse.

Table 4.13. False negative rate (FNR) metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.035	0.059	0.036	0.018	0.020	0.019	0.039
20	0.054	0.023	0.022	0.023	0.038	0.024	0.023
40	0.051	0.018	0.019	0.023	0.017	0.019	0.020
60	0.016	0.018	0.019	0.035	0.019	0.020	0.016

End of the Table 4.13

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
80	0.015	0.016	0.017	0.018	0.021	0.017	0.017
100	0.019	0.017	0.014	0.015	0.017	0.017	0.016
200	0.014	0.017	0.025	0.031	0.016	0.014	0.016
400	0.012	0.019	0.016	0.012	0.012	0.010	0.013
600	0.009	0.008	0.010	0.010	0.013	0.011	0.011
800	0.006	0.007	0.004	0.006	0.010	0.009	0.011
1000	0.003	0.003	0.006	0.005	0.007	0.007	0.007

4.5.2.5. Error rate

Error rate values are presented in Table 4.15. It is consistently seen that the 600 system calls sequence is the first one that generates significantly better results than the smaller ones.

Table 4.14. Comparison of 95% confidence intervals for an error rate of FCN and AWSCTD family models (1000 foremost system calls)

Model	Interval
AWSCTD-CNN-S	[0.002,0.012]
AWSCTD-CNN-D	[0.002,0.012]
AWSCTD-CNN-LSTM	[0.003,0.015]
AWSCTD-CNN-GRU	[0.002,0.014]
FCN	[0.004,0.018]
LSTM-FCN	[0.004,0.018]
GRU-FCN	[0.005,0.019]

The best results (the smallest number of error rate) equal to 0.007 were obtained by static (AWSCTD-CNN-S) and dynamic (AWSCTD-CNN-D) models. The best value of the FCN family was equal to 0.011 for FCN and LSTM-FCN models. The differences between families are of no significant statistical difference and do not demonstrate the superiority of dual-flow models.

Table 4.14 shows 95% confidence interval values of the error rate for all tested complicated models (1000 system calls). There is a 95% likelihood that the

average confidence interval [0.002, 0.012] covers the true classification error of the best performing AWSCTD-CNN-S and AWSCTD-CNN-D models on unseen data. Furthermore, the lower bound of interval (0.002) belongs to the AWSCTD family models and are two times bigger for the FCN family.

Table 4.15. Error rate metric values

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.112	0.120	0.114	0.110	0.107	0.108	0.112
20	0.083	0.073	0.073	0.072	0.077	0.073	0.072
40	0.074	0.062	0.061	0.064	0.064	0.065	0.065
60	0.059	0.058	0.059	0.066	0.060	0.062	0.061
80	0.057	0.057	0.057	0.058	0.062	0.059	0.059
100	0.058	0.056	0.056	0.056	0.056	0.057	0.057
200	0.053	0.055	0.056	0.059	0.052	0.054	0.053
400	0.048	0.050	0.050	0.047	0.050	0.048	0.050
600	0.015	0.014	0.017	0.015	0.017	0.017	0.017
800	0.011	0.011	0.011	0.011	0.014	0.014	0.017
1000	0.007	0.007	0.009	0.008	0.011	0.011	0.012

4.5.2.6. The Training Time

As it can be seen from Table 4.20, the static AWSCTD-CNN-S model demonstrates the fastest training time for the 1000 system calls sequence. The next model by speed is FCN, which is slower by around 21%. All other models were slower than AWSCTD-CNN-S, with results varying from 64–68% (Figure 4.22).

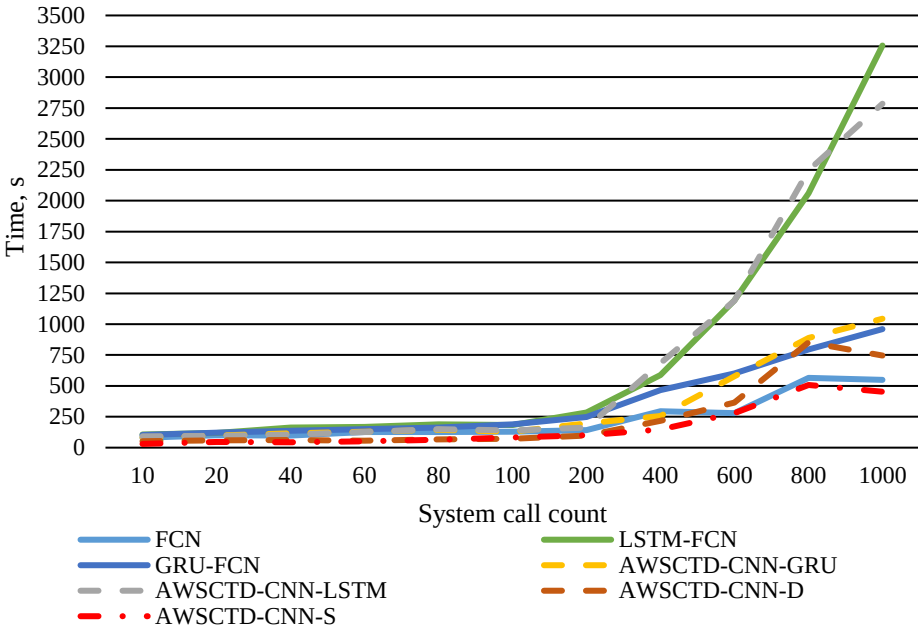


Fig. 4.22. Training time for one-fold

The training times (Table A.5) are given for one-fold. The differentiation in model speeds starts from the sequence of 200 system calls (Figure 4.20).

The worst training time results were obtained for the LSTM-FCN and AWSCTD-CNN-LSTM models. This can be explained by the number of parameters of these models.

4.5.2.7. The Testing Time

The testing time is defined as the time needed to perform the classification of one sample. LSTM-based models demonstrate similar behaviour, as in the case of training experiments, and require three times more time to perform the detection compared to the best performing AWSCTD-CNN-S model, which retained the leadership, as well as in training task. The testing times for all models and sequence lengths are provided in Table A.6.

It can be seen from Figure 4.17 that static models are more suitable not only for training but also for classification of long univariate times series data. In our case, it would be long (> 400) system call sequences.

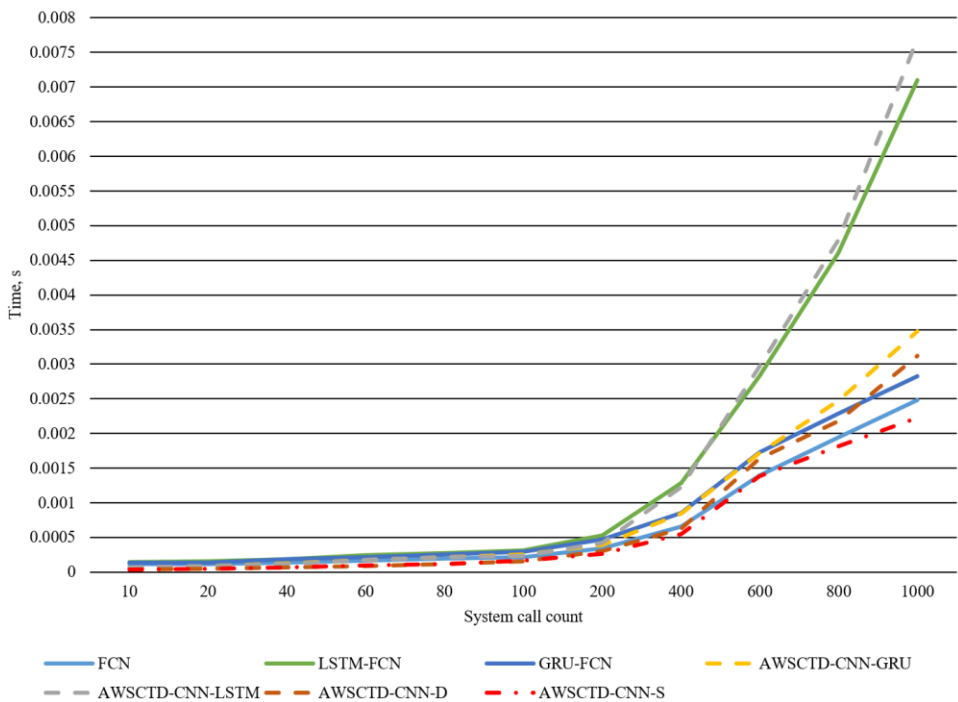


Fig. 4.23. Classification time for one sample

Since classification time is crucial for attack and malware detection in a real-time application, which could suggest that the best performing AWSCTD-CNN-S is the best suitable candidate for the defined task, despite its relative simplicity.

4.5.2.8. The Comparison of the Number of Epochs Needed to Reach the Saturation

The comparison of epochs required to reach the saturation during the training process has demonstrated the correlation to the training time. The relation of accuracy versus epochs for all models presented in Figure 4.24 and Figure 4.25 for the sequence of 1000 system calls gives the best accuracy.

The AWSCTD-CNN-S, AWSCTD-CNN-D, and AWSCTD-CNN-LSTM models reached the saturation in maximum of 25 epochs, and they demonstrate better stability. The AWSCTD-CNN-GRU and FCN models required 30 epochs. The maximum number of epochs required by the LSTM-FCN and GRU-FCN models was 30 or more epochs.

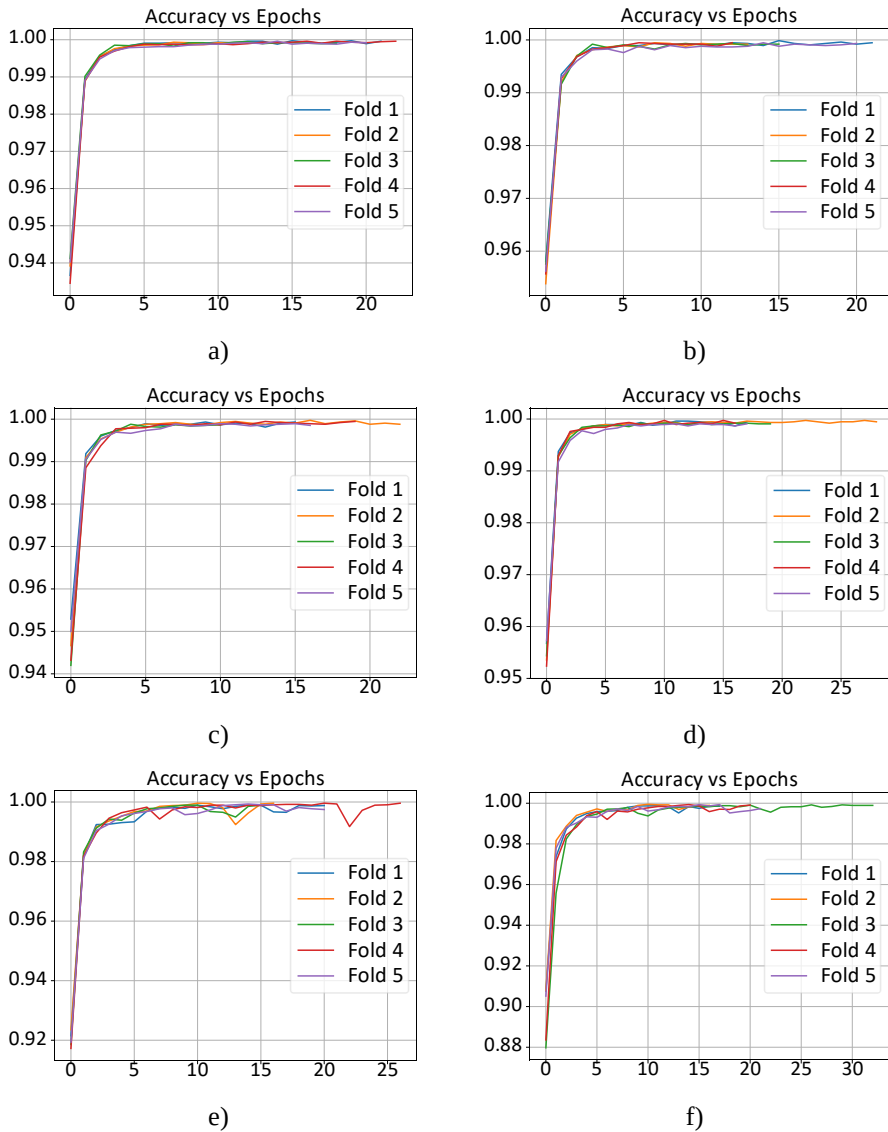


Fig. 4.24. Number of epochs vs accuracy for the 1000 system call sequence: a) AWS-CTD-CNN-S; b) AWSCTD-CNN-D; c) AWSCTD-CNN-LSTM; d) AWSCTD-CNN-GRU; e) FCN; f) LSTM-FCN

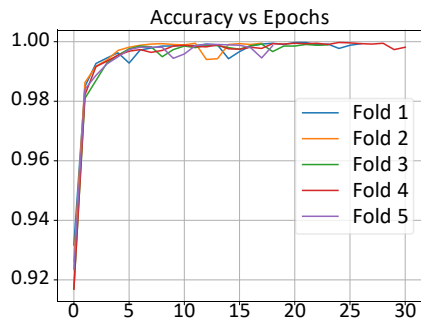


Fig. 4.25. Number of epochs vs accuracy for the 1000 system call sequence (GRU-FCN model)

4.5.2.9. Comparison of the CMs

The values of CMs for 1000 system calls (Figure 4.26 and Figure 4.27) reveal that the lowest intrusion misclassification values are obtained by static (AWSCTD-CNN-S) and dynamic (AWSCTD-CNN-D) models and are equal to 26 and 25, respectively. The FCN family models generate two times more misclassified results: 53, 55, and 57, respectively, for FCN, LSTM-FCN, and GRU-FCN. The AWSCTD-CNN-GRU model shows the best misclassification value for benign applications, where only 32 items have been identified as an intrusion.

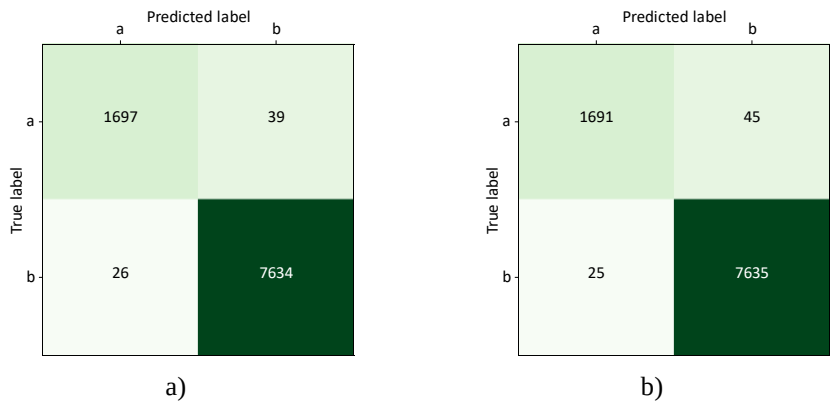


Fig. 4.26. The CM results for 1000 system calls: a) AWSCTD-CNN-S; b) AWSCTD-CNN-D

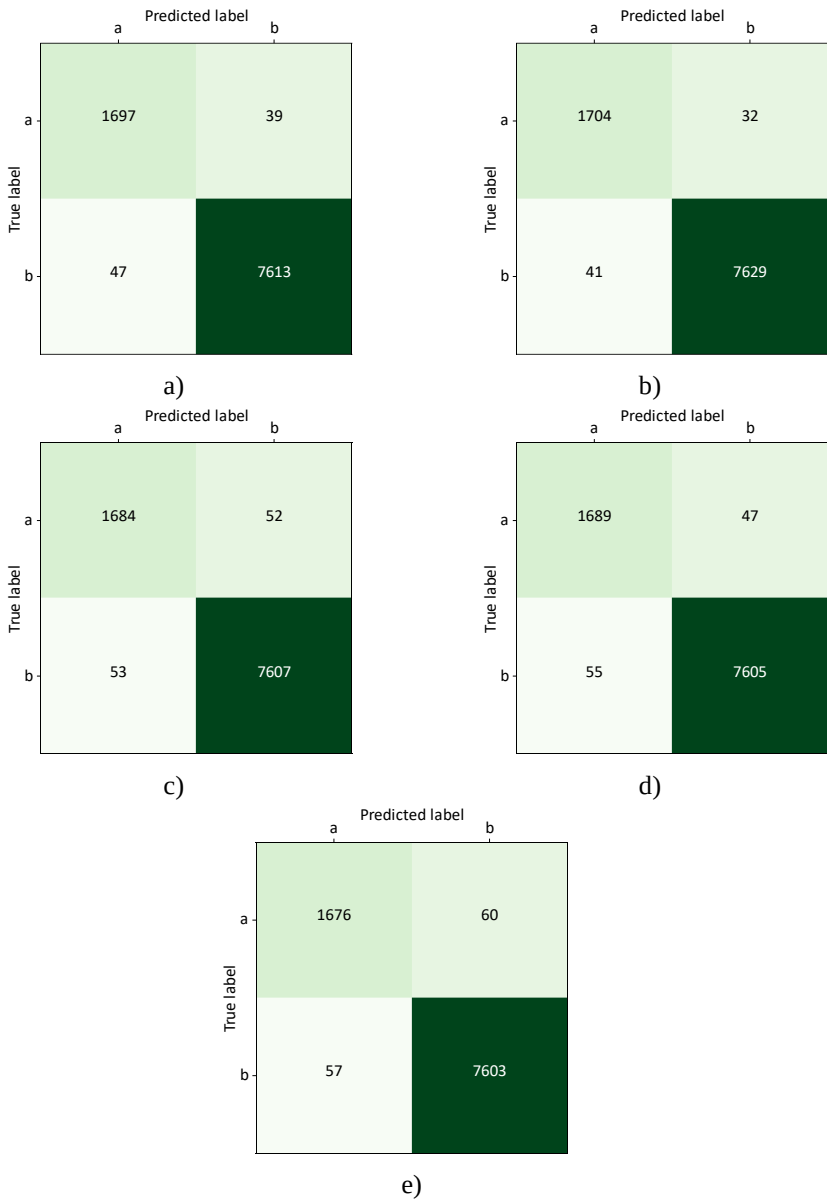


Fig. 4.27. The CM results for 1000 system calls: a) AWSCTD-CNN-LSTM; b) AWSCTD-CNN-GRU; c) FCN; d) LSTM-FCN; e) GRU-FCN

Although AWSCTD-CNN-GRU demonstrated the best result for one specific task, AWSCTD-CNN-S can still be considered as the overall winner when taking into consideration all previously discussed metrics.

4.5.2.10. Saturation of the Accuracy

To check the number of system calls required to achieve accuracy saturation the next experiment has been conducted. The system calls sequences of the following length has been used in that experiment: 10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900 and 2000. It has been prepared sequences till 3000 system calls, but computer used in the experiment was unable to process such a big amount of data and memory errors started to appear on the sequence of 2100 system calls.

The results of the experiment are provided in Figure 4.28. The best performing model of AWSCTD-CNN-S has been used. It can be seen that the sequence of 1000 system calls is representing a value of saturated accuracy. None of the following sequences was capable of improving the accuracy of 99.3% in a statistically significant way – it varied from 99.4% to 99.5%.

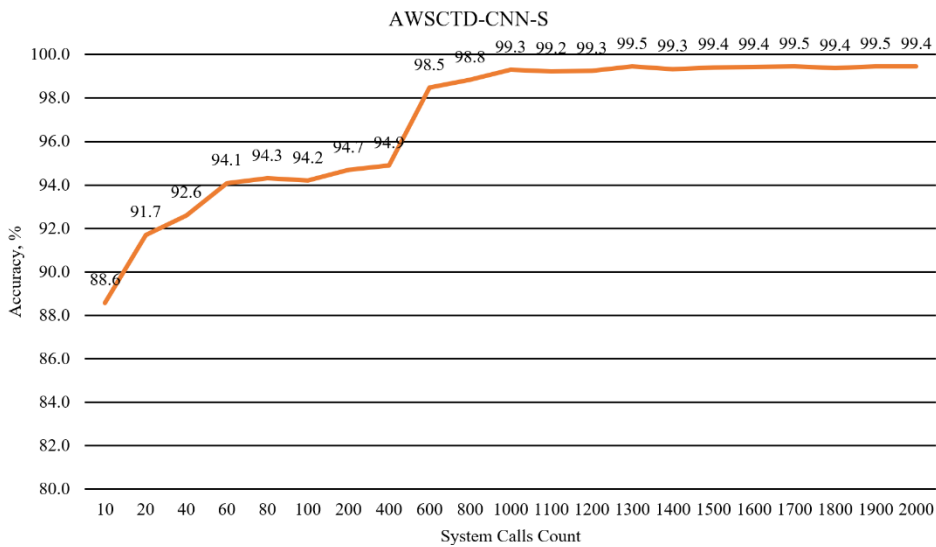


Fig. 4.28. The results of the accuracy in the saturation experiment

The comparison of confusion matrixes of 1000 and 1900 system calls (Figure 4.29) is demonstrating that a longer sequence is producing a smaller value of false positives and false negatives.

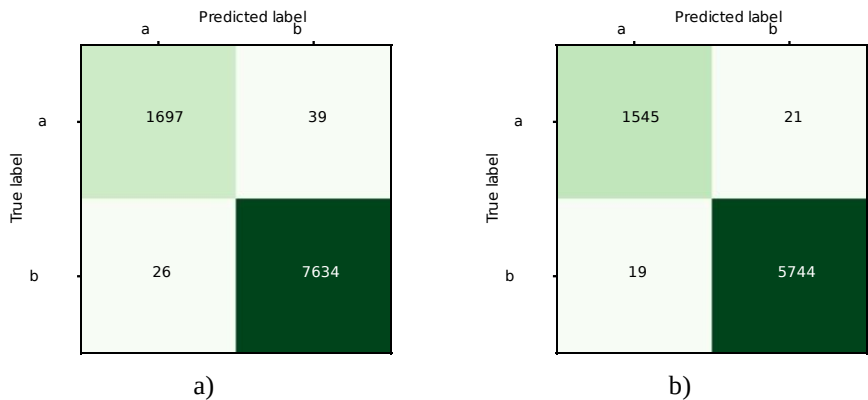


Fig. 4.29. The comparison of confusion matrixes between 1000 and 1900 system calls sequences: a) 1000 system calls; b) 1900 system calls. Class labels: Benign (a); Malware (b)

The FPR and FNR values of 1900 system calls are the following: 0.013 and 0.003, respectively. The FPR and FNR values of 1000 system calls were the following: 0.022 and 0.007, respectively. It is clear that despite the fact of stalling accuracy – FPR and PNR values are better for 1900 system calls.

4.6. Conclusions of Chapter 4

1. A comparative analysis of vanilla DL methods, including LSTM, GRU, and CNN was performed in order to evaluate their efficiency for attack classification as well as their ability to distinguish malicious and benign activity. The analysis was performed on the AWSCTD dataset, which includes 112.56 million system calls from 12110 executable malware samples and was expanded with 3145 benign software samples with 16.3 million system calls. The application of such a set has increased the classification and intrusion identification accuracy even with vanilla models by 13–38%, compared to the results achieved by other researchers. In general, the achieved accuracy of over 90% allows the application of DL techniques in hybrid or enterprise-oriented security solutions that combine automatic detection of the major part of anomalies leaving unclear cases for human-expert analysis.

2. It was determined that all three vanilla-type DL models have reached higher than 90% accuracy while solving a malware classification task with a sequence of 80 system calls. All three models generated improved results over simple NN and SVM models on larger data samples, while the latter demonstrates considerably better training times. The best results were obtained with CNNs with up to 93.9% (MAE of 0.03) accuracy while performing the family classification task and 99.3% (MAE of 0.01) rate while solving intrusion detection task. CNN outperforms LSTM and GRU models in all the cases. CNN also shows the best results as compared to the EER values of DL methods used. A system calls minimization technique had a positive influence on all results. The increase of sequence length for vanilla DL models has resulted in an exponential increase of the training time and a reverse dependency of the number of epochs before saturation.
3. In total, seven advanced DL models were tested: dual-flow LSTM-FCN and GRU-FCN state-of-the-art models that were proved as the most efficient in tests on many datasets; new synthetic dual-flow AWSCTD-CNN-LSTM and AWSCTD-CNN-GRU models, combining CNN and RNN flows; earlier defined single-flow FCN and AWSCTD-CNN-D models; and the newly introduced static single-flow AWSCTD-CNN-S model.
4. The experiments performed with AWSCTD dataset, have not demonstrated any advantages of dual-flow DL models for the host-level intrusion and malware detection task in comparison with the single-flow models. The reached accuracy of dual-flow models was comparable (99.2% and varying MAE from 0.01 to 0.02) with more vanilla single-flow models (99.3% with constant MAE of 0.01), taking into account the 1% error rate in case of a sequence of 1000 of system calls. The performance metrics results were somewhat identical in their accuracy. In training and testing time metrics, the results for dual-flow models were 3- to 7-fold slower compared to the best performing single flow model (which are crucial for the intrusion and malware detection tasks).
5. All advanced DL models tested were able to reach 90% accuracy with as little as 20 system calls and had practically applicable accuracy of 98% with a sequence of 600 system calls. The relatively best FPR value (approximately 2%) was reached by the AWSCTD-CNN-GRU, while the best FNR value was reached by AWSCTD-CNN-S and AWSCTD-CNN-D (less than 1%). Taking into account all of the evaluation metrics, the newly tested AWSCTD-CNN-S can be considered as the overall winner, having the best accuracy (99.3% with MAE of 0.01), shortest training and testing times, and the best FNR rate.

General Conclusions

1. The scientific literature review performed has demonstrated the importance of anomaly-based methods for the host-level intrusion and malware detection task, since the classical detection approach based on signatures has a number of flaws, such as inability to detect zero-day and evolved attacks. Current scientific research concentrates on the application of ML/DL methods but is constrained by the lack of exhaustive datasets, there are no high-quality datasets for Windows OS, that could be used for methods training. The research has also demonstrated that sequences of system calls invoked by malware and exploits used for the attack on the system level are seen as one of the most perspective sources for intrusion and malware detection. The analysis performed has allowed clarifying that research on improvement of anomaly-based host-level intrusion and malware detection by applying ML and DL methods should be concentrated on the creation of exhaustive dataset of system calls sequences by malicious and benign applications with further application of the created dataset for training the most appropriate ML/DL models.
2. The proposed method for host-level dataset generation is based on malware execution in a sterile, isolated virtual machine environment with further anomaly activity collection. The method was implemented and tested only with free or open-source tools and freely available malware samples.

The tests performed have proved the method stability and method suitability for host-level anomaly dataset generation. Automated anomaly generation allows flexible training data-set expansion, response to the new attack types and generation of specific on-demand datasets. No interruptions or errors related to the malware execution or escape were noticed. AWSCTD was generated for 10 276 malware samples. The dataset size exceeds the size of previously known datasets by order and includes a much more comprehensive representation of malware types and system calls (system call arguments list and return value) that allow more in-depth HIDS training. Later AWSCTD was expanded with 16.3 million system calls by benign applications. The introduction of a new exhaustive dataset provides a possibility for further practical and scientific applications of data analysis methods in host-level intrusion detection.

3. ML methods were utilized to demonstrate the applicability of malware generated systems calls sequences for IDS reported alerts prioritization use. 7 types of ML methods were applied to classify malware by family. The results have shown that this approach could be used in practical applications – the best detection rate of 92.4% with MAE of 0.03 was obtained by using SVM with the 100 features vector. Comparable results were achieved by J48 and LMT methods. They generated 92.1% (MAE of 0.05) classification results. J48 has also shown the best training and testing times, that is very useful when new samples must be introduced for training. Despite of achieved high accuracy rates reaching 90–92% (suitable for some practical applications when additional analysis is performed by external human experts) and relatively low training time needed for ML methods, further research on DL methods applicability is needed in order to minimize the FNR.
4. A comparative analysis of vanilla DL methods was performed in order to evaluate their efficiency for attack classification as well as their ability to distinguish malicious and benign activity. The application of AWSCTD has increased the classification and intrusion identification accuracy even with vanilla models by 13–38%, compared to the results achieved by other researchers. The achieved accuracy of over 90% allows the application of DL techniques in hybrid or enterprise-oriented security solutions that combine automatic detection of the major part of anomalies leaving unclear cases for human-expert analysis. All three models generated improved results over simple NN and SVM models on larger data samples, while the latter demonstrates considerably better training times. The best results were obtained with CNNs with up to 93.0% accuracy (MAE of 0.03) while performing family classification task and 99.3% rate (MAE

of 0.01) while solving intrusion detection task. CNN outperformed sequence-based LSTM and GRU models in all the cases. A system calls minimization technique had a positive influence on all results. The increase of sequence length for vanilla DL models has resulted in an exponential increase of the model training time. A reverse dependency of the number of epochs before saturation on the system call sequence length was determined.

5. The experiments with seven advanced DL models have not demonstrated any advantages of dual-flow DL models for the host-level intrusion and malware detection task in comparison with the single-flow models. The reached accuracy of dual-flow models was comparable (99.2% and varying MAE from 0.01 to 0.02) with more vanilla single-flow models (99.3% with constant MAE of 0.01 and 1% error rate), in case of a sequence of 1000 of system calls. The results of performance metrics were somewhat identical in their accuracy but were losing the competition in the training and testing time metrics: dual-flow models were 3- to 7-fold slower compared to the best performing single flow model, consuming more computing resources, and requiring more epochs to reach the training saturation. All tested advanced DL models were able to reach 90% accuracy with as little as 20 system calls and had practically applicable accuracy of 98% with a sequence of 600 system calls. The relatively best FPR value (approximately 2%) was reached by the AWSCTD-CNN-GRU, while the best FNR value was reached by AWSCTD-CNN-S and AWSCTD-CNN-D models and was equal to less than 1%. The newly proposed AWSCTD-CNN-S can be considered as the overall winner, having the best accuracy (99.3%), shortest training and testing times, and the best FNR rate. Such high classification accuracy allows using a winner model in security solutions supported by human-experts with 600 foremost system calls and completely automated application in commercial IDS and AV systems when 1000 system calls are analysed.

References

- Abad, C., Taylor, J., Sengul, C., Yurcik, W., Yuanyuan Zhou, & Rowe, K. (2003). Log correlation for intrusion detection: a proof of concept. *19th Annual Computer Security Applications Conference, 2003. Proceedings., 2003-Janua(Acsac)*, 255–264. <https://doi.org/10.1109/CSAC.2003.1254330>
- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., & others. (2016). Tensorflow: A System for Large-Scale Machine Learning. *Osdi*, 16, 265–283.
- Agrawal, S., & Agrawal, J. (2015). Survey on anomaly detection using data mining techniques. *Procedia Computer Science*. <https://doi.org/10.1016/j.procs.2015.08.220>
- Ahmad, I, Swati, S. U., & Mohsin, S. (2007). Intrusions detection mechanism by resilient back propagation (RPROP). *European Journal of Scientific Research*, 17(4), 523–530. <http://www.scopus.com/inward/record.url?eid=2-s2.0-34547863151&partnerID=40&md5=4a6eed8c0eef85943021c37f3884f783>
- Ahmad, Iftikhar, Abdullah, A. B., Alghamdi, A. S., Baykara, N. a, & Mastorakis, N. E. (2009). Artificial neural network approaches to intrusion detection: a review. In *Telecommunications and Informatics* (pp. 200–205).
- Ahmad, Iftikhar, Ansari, M. A., & Mohsin, S. (2008). Performance Comparison between Backpropagation Algorithms Applied to Intrusion Detection in Computer Network Systems. *International Conference on NEURAL NETWORKS*, 231–236.

- Ahmed, Y. A., Koçer, B., Huda, S., Saleh Al-rimy, B. A., & Hassan, M. M. (2020). A system call refinement-based enhanced Minimum Redundancy Maximum Relevance method for ransomware early detection. *Journal of Network and Computer Applications*, 167(January), 102753. <https://doi.org/10.1016/j.jnca.2020.102753>
- Ait Tchakoucht, T., Ezziyyani, M., Jbilou, M., & Salaun, M. (2015). Behavioral approach for intrusion detection. *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, 2016-July, 1–5. <https://doi.org/10.1109/AICCSA.2015.7507118>
- Al Jallad, K., Aljnidi, M., & Desouki, M. S. (2020). Anomaly detection optimization using big data and deep learning to reduce false-positive. *Journal of Big Data*, 7(1), 68. <https://doi.org/10.1186/s40537-020-00346-1>
- Aldweesh, A., Derhab, A., & Emam, A. Z. (2020). Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems*, 189, 105124. <https://doi.org/10.1016/j.knsys.2019.105124>
- Alsubhi, K., Al-Shaer, E., & Boutaba, R. (2008). Alert prioritization in Intrusion Detection Systems. *NOMS 2008 - IEEE/IFIP Network Operations and Management Symposium: Pervasive Management for Ubiquitous Networks and Services*, 33–40. <https://doi.org/10.1109/NOMS.2008.4575114>
- Alsulami, B., Srinivasan, A., Dong, H., & Mancoridis, S. (2017). Lightweight behavioral malware detection for windows platforms. *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, 2018-Janua, 75–81. <https://doi.org/10.1109/MALWARE.2017.8323959>
- Amini, M., & Jalili, R. (2004). Network-Based Intrusion Detection Using Unsupervised Adaptive Resonance Theory (ART) *. *System*.
- Anderson, H. S., & Roth, P. (2018). *EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models*. <http://arxiv.org/abs/1804.04637>
- Anderson, J. P. (1980). Computer security threat monitoring and surveillance. *Technical Report James P Anderson Co Fort Washington Pa*. <https://doi.org/citeulike-article-id:592588>
- Annachhatre, C., Austin, T. H., & Stamp, M. (2015). Hidden Markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, 11(2), 59–73. <https://doi.org/10.1007/s11416-014-0215-x>
- Apap, F., Honig, A., Hershkop, S., Eskin, E., & Stolfo, S. (2002). *Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses* (pp. 36–53). https://doi.org/10.1007/3-540-36084-0_3
- Auskalnis, J., Paulauskas, N., & Baskys, A. (2018). Application of Local Outlier Factor Algorithm to Detect Anomalies in Computer Network. *Elektronika Ir Elektrotechnika*, 24(3), 96–99. <https://doi.org/10.5755/j01.eie.24.3.20972>

- Axelsson, S. (2000). The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security*, 3(3), 186–205. <https://doi.org/10.1145/357830.357849>
- Azad, C., & Jha, V. K. (2013). Data Mining in Intrusion Detection: A Comparative Study of Methods, Types and Data Sets. *International Journal of Information Technology and Computer Science*, 5(8), 75–90. <https://doi.org/10.5815/ijitcs.2013.08.08>
- Azeez, Ayemobola, Misra, Maskeliūnas, & Damaševičius. (2019). Network Intrusion Detection with a Hashing Based Apriori Algorithm Using Hadoop MapReduce. *Computers*, 8(4), 86. <https://doi.org/10.3390/computers8040086>
- Azeez, N. A., Salaudeen, B. B., Misra, S., Damaševičius, R., & Maskeliūnas, R. (2020). Identifying phishing attacks in communication networks using URL consistency features. *International Journal of Electronic Security and Digital Forensics*, 12(2), 200. <https://doi.org/10.1504/IJESDF.2020.106318>
- Bace, R., & Mell, P. (2001). NIST special publication on intrusion detection systems. In *Nist Special Publication*. [https://doi.org/10.1016/S1361-3723\(01\)00614-5](https://doi.org/10.1016/S1361-3723(01)00614-5)
- Bahdanau, D., Cho, K. H., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.
- Berkhin, P. (2006). A Survey of Clustering Data Mining Techniques. In *Grouping Multidimensional Data* (Issue c, pp. 25–71). Springer-Verlag. https://doi.org/10.1007/3-540-28349-8_2
- Berlin, K., Slater, D., & Saxe, J. (2015). Malicious Behavior Detection using Windows Audit Logs. *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security – AISec '15*, 35–44. <https://doi.org/10.1145/2808769.2808773>
- Bhattacharyya, D. K., & Kalita, J. K. (2013). *Network Anomaly Detection: A Machine Learning Perspective*. Chapman and Hall/CRC.
- Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2015). Towards generating real-life datasets for network intrusion detection. *International Journal of Network Security*, 17(6), 683–701.
- Bivens, A., Palagiri, C., Smith, R., Szymanski, B., & Embrechts, M. (2002). Network-based intrusion detection using neural networks. *Neural Networks*, 12, 579–584. <http://assassin.cs.rpi.edu/~szymansk/papers/annie02.pdf>
- Boughorbel, S., Jarray, F., & El-Anbari, M. (2017). Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. *PLoS ONE*, 12(6), 1–17. <https://doi.org/10.1371/journal.pone.0177678>
- Bouzar-Benlabiod, L., Rubin, S. H., Belaidi, K., & Haddar, N. E. (2020). RNN-VED for Reducing False Positive Alerts in Host-based Anomaly Detection Systems. *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, 17–24. <https://doi.org/10.1109/IRI49571.2020.00011>

- Brugger, T. (2007). *KDD Cup'99 dataset (Network Intrusion) considered harmful*. KDNuggets Newsletter. <https://www.kdnuggets.com/news/2007/n18/4i.html>
- Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, 18(2). <https://doi.org/10.1109/COMST.2015.2494502>
- Bulavas, V. (2018). Investigation of network intrusion detection using data visualization methods. *59th International Scientific Conference on Information Technology and Management Science of Riga Technical University, ITMS 2018 - Proceedings*. <https://doi.org/10.1109/ITMS.2018.8552977>
- Butun, I., Morgera, S. D., & Sankar, R. (2014). A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *IEEE Communications Surveys Tutorials*, 16(1), 266–282.
- Camacho, J., Pérez-Villegas, A., Garcíá-Teodoro, P., & MacÍá-Fernández, G. (2016). PCA-based multivariate statistical network monitoring for anomaly detection. *Computers and Security*, 59, 118–137. <https://doi.org/10.1016/j.cose.2016.02.008>
- Cannady, J. D. (1998). Artificial neural networks for misuse detection. *Proceedings of the 21st National Information Systems Security Conference*, 368–381. <https://doi.org/citeulike-article-id:9827770>
- Canzanese, R., Mancoridis, S., & Kam, M. (2015). System Call-Based Detection of Malicious Processes. *Proceedings - 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015*, 119–124. <https://doi.org/10.1109/QRS.2015.26>
- Canzanese, R., Mancoridis, S., & Kam, M. (2016). Run-time classification of malicious processes using system call analysis. *2015 10th International Conference on Malicious and Unwanted Software, MALWARE 2015*. <https://doi.org/10.1109/MALWARE.2015.7413681>
- Catak, F. O., & Yazı, A. F. (2019). *A Benchmark API Call Dataset for Windows PE Malware Classification*. May, 1–13. <http://arxiv.org/abs/1905.01999>
- Chaibi, N., Atmani, B., & Mokaddem, M. (2020). Deep Learning Approaches to Intrusion Detection: A new Performance of ANN and RNN on NSL-KDD. *Proceedings of the 1st International Conference on Intelligent Systems and Pattern Recognition*, 45–49. <https://doi.org/10.1145/3432867.3432889>
- Chakir, E., Moughit, M., & Khamlichi, Y. I. (2018). *Building an Efficient Alert Management Model for Intrusion Detection Systems*. 3(1), 18–24.
- Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection. *ACM Computing Surveys*, 41(3), 1–58. <https://doi.org/10.1145/1541880.1541882>
- Chawla, A., Lee, B., Fallon, S., & Jacob, P. (2018). Host based Intrusion Detection System with Combined CNN/RNN Model. *Proceedings of Second International Workshop on AI in Security*.

- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*. <https://doi.org/10.1613/jair.953>
- Chen, C. M., Guan, D. J., Huang, Y. Z., & Ou, Y. H. (2016). Anomaly network intrusion detection using Hidden Markov Model. *International Journal of Innovative Computing, Information and Control*, 12(2), 569–580. <https://doi.org/10.1109/ICSMB.2010.5735410>
- Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. <http://arxiv.org/abs/1409.1259>
- Chollet, F., & others. (2015). *Keras*. <https://keras.io>
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *ArXiv Preprint ArXiv:1412.3555*. <https://doi.org/1412.3555v1>
- Connor, J., & Atlas, L. (1991). *Recurrent Neural Networks and Time Series Prediction*. 301–306.
- Creech, G., & Hu, J. (2013). Generation of a new IDS test dataset: Time to retire the KDD collection. *IEEE Wireless Communications and Networking Conference, WCNC*, 4487–4492. <https://doi.org/10.1109/WCNC.2013.6555301>
- Creech, G., & Hu, J. (2014). A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. *IEEE Transactions on Computers*, 63(4), 807–819. <https://doi.org/10.1109/TC.2013.13>
- Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd International Conference on Machine Learning - ICML '06*, 148, 233–240. <https://doi.org/10.1145/1143844.1143874>
- Dawoud, A., Shahristani, S., & Raun, C. (2018). Deep Learning for Network Anomalies Detection. *2018 International Conference on Machine Learning and Data Engineering (ICMLDE)*, 149–153. <https://doi.org/10.1109/ICMLDE.2018.00035>
- De Teyou, G. K., & Ziazet, J. (2019). *Convolutional Neural Network for Intrusion Detection System In Cyber Physical Systems*. <http://arxiv.org/abs/1905.03168>
- Deng, L., & Platt, J. C. (2014). Ensemble deep learning for speech recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH, September*, 1915–1919.
- Deng, L., & Yu, D. (2013). Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, 7(3–4), 197–387. <https://doi.org/10.1136/bmj.319.7209.0a>
- Denning, D. E. (1987). An Intrusion-Detection Model. *Ieee Transactions on Software Engineering*, 13(2), 222–232. <https://doi.org/10.1109/TSE.1987.232894>
- Dua, S., & Du, X. (2016). Data Mining and Machine Learning in Cybersecurity. In *Data Mining and Machine Learning in Cybersecurity*. Auerbach Publications. <https://doi.org/10.1201/b10867>

- Dumoulin, V., & Visin, F. (2016). *A guide to convolution arithmetic for deep learning*. 1–28. <https://doi.org/10.1051/0004-6361/201527329>
- DynamoRIO. (2020). *Dr. Memory Memory Debugger for Windows and Linux*. <http://www.drmemory.org/>
- Elhag, S., Fernández, A., Bawakid, A., Alshomrani, S., & Herrera, F. (2015). On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on Intrusion Detection Systems. *Expert Systems with Applications*. <https://doi.org/10.1016/j.eswa.2014.08.002>
- Elsayed, N., Maida, A. S., & Bayoumi, M. (2019). Deep gated recurrent and convolutional network hybrid model for univariate time series classification. *International Journal of Advanced Computer Science and Applications*, 10(5), 654–664. <https://doi.org/10.14569/ijacsa.2019.0100582>
- Fawcett, T. (2005). An introduction to ROC analysis. *Irbm*, 35(6), 299–309. <https://doi.org/10.1016/j.patrec.2005.10.010>
- Fiore, U., Palmieri, F., Castiglione, A., & De Santis, A. (2013). Network anomaly detection with the restricted Boltzmann machine. *Neurocomputing*, 122, 13–23. <https://doi.org/10.1016/j.neucom.2012.11.050>
- Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for Unix processes. *Proceedings 1996 IEEE Symposium on Security and Privacy*, 120–128. <https://doi.org/10.1109/SECPRI.1996.502675>
- Forrest, Stephanie, Hofmeyr, S. A., & Somayaji, A. (1998). Intrusion Detection using Sequences of System Calls (1998). *Journal of Computer Security*, Volume: 65(Issue:), Pages: 43-59.
- Gao, N., Gao, L., Gao, Q., & Wang, H. (2014). An Intrusion Detection Model Based on Deep Belief Networks. *2014 Second International Conference on Advanced Cloud and Big Data*, 247–252. <https://doi.org/10.1109/CBD.2014.41>
- Ghosh, A. K., Schwartzbard, A., & Schatz, M. (1999). Learning program behavior profiles for intrusion detection. *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring.*, 51–62. <https://doi.org/10.1109/USENIX.1999.81880446375>
- Gibert, D. (2016). Convolutional Neural Networks for Malware Classification. In *University Rovira i Virgili, Tarragona, Spain*.
- Goldberg, D. E., & Holland, J. H. (1988). Genetic Algorithms and Machine Learning. In *Machine Learning*. <https://doi.org/10.1023/A:1022602019183>
- Gonda, O. (2014). Understanding the threat to SCADA networks. *Network Security*. [https://doi.org/10.1016/S1353-4858\(14\)70093-5](https://doi.org/10.1016/S1353-4858(14)70093-5)
- Graham, B. (2014). *Fractional Max-Pooling*. 1–10. <http://arxiv.org/abs/1412.6071>
- Griffin, J., Pennington, A., & Bucy, J. (2003). *On the feasibility of intrusion detection inside workstation disks*. 1–27.

- Grimmer, M., Röhling, M. M., Kricke, M., Franczyk, B., & Rahm, E. (2018). Intrusion Detection on System Call Graphs. *Proceedings - 25. DFN-Konferenz "Sicherheit in Vernetzten Systemen,"* 1–18.
- Guo, Y., Liu, Y., Bakker, E. M., Guo, Y., & Lew, M. S. (2018). CNN-RNN: a large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, 77(8), 10251–10271. <https://doi.org/10.1007/s11042-017-5443-x>
- Gupta, S., & Kumar, P. (2015). An Immediate System Call Sequence Based Approach for Detecting Malicious Program Executions in Cloud Environment. *Wireless Personal Communications*, 81(1), 405–425. <https://doi.org/10.1007/s11277-014-2136-x>
- Hafsa, M., & Jemili, F. (2018). Comparative Study between Big Data Analysis Techniques in Intrusion Detection. *Big Data and Cognitive Computing*, 3(1), 1. <https://doi.org/10.3390/bdcc3010001>
- Haider, W., Hu, J., Slay, J., Turnbull, B. P., & Xie, Y. (2017). Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*, 87(November 2016), 185–192. <https://doi.org/10.1016/j.jnca.2017.03.018>
- Haider, Waqas, Creech, G., Xie, Y., & Hu, J. (2016). Windows based data sets for evaluation of robustness of Host based Intrusion Detection Systems (IDS) to zero-day and stealth attacks. *Future Internet*, 8(3). <https://doi.org/10.3390/fi8030029>
- Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., & Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. In *Expert Systems with Applications*. <https://doi.org/10.1016/j.eswa.2016.12.035>
- Hay, A., Cid, D., Bary, R., & Northcutt, S. (2008). System Integrity Check and Rootkit Detection. In *OSSEC Host-Based Intrusion Detection Guide* (pp. 149–174). Elsevier. <https://doi.org/10.1016/B978-1-59749-240-9.00005-3>
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. <https://doi.org/10.1109/CVPR.2016.90>
- Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., & Wolber, D. D. (1990). A network security monitor. *Proceedings of the Symposium on Security and Privacy*, 296–304. <https://doi.org/10.1109/risp.1990.63859>
- Heller, K., Svore, K., Keromytis, A. D., & Stolfo, S. (2003). One class support vector machines for detecting anomalous windows registry accesses. *Workshop on Data Mining for Computer Security (DMSEC), Melbourne, FL, November 19, 2003*.
- Hinton, G. (2009). Deep belief networks. *Scholarpedia*. <https://doi.org/10.4249/scholarpedia.5947>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>

- Hu, J., Yu, X., Qiu, D., & Chen, H.-H. (2009). A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection. *IEEE Network*, 23(1), 42–47. <https://doi.org/10.1109/MNET.2009.4804323>
- Hwang, R. H., Peng, M. C., Nguyen, V. L., & Chang, Y. L. (2019). An LSTM-based deep learning approach for classifying malicious traffic at the packet level. *Applied Sciences (Switzerland)*, 9(16). <https://doi.org/10.3390/app9163414>
- Japertas, S., & Baksys, T. (2018). Method of Early Staged Cyber Attacks Detection in IT and Telecommunication Networks. *Elektronika Ir Elektrotechnika*, 24(3), 68–77. <https://doi.org/10.5755/j01.eie.24.3.20981>
- Jewell, B., & Beaver, J. (2011). Host-based data exfiltration detection via system call sequences. *6th International Conference on Information Warfare and Security, ICIW 2011*, 134–142.
- Ji, S., Kim, J., & Im, H. (2019). A comparative study of bitcoin price prediction using deep learning. *Mathematics*, 7(10). <https://doi.org/10.3390/math7100898>
- Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, & Li Fei-Fei. (2009). *ImageNet: A large-scale hierarchical image database*. <https://doi.org/10.1109/cvprw.2009.5206848>
- Jin Kim, Nara Shin, Jo, S. Y., & Sang Hyun Kim. (2017). Method of intrusion detection using deep neural network. *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 313–316. <https://doi.org/10.1109/BIGCOMP.2017.7881684>
- Jirapummin, C., Wattanapongsakorn, N., & Kanthamanon, P. (2002). Hybrid Neural Networks for Intrusion Detection System. *Network*, 2–5.
- Joo, D., Hong, T., & Han, I. (2003). The neural network models for IDS based on the asymmetric costs of false negative errors and false positive errors. *Expert Systems with Applications*, 25(1), 69–75. [https://doi.org/10.1016/S0957-4174\(03\)00007-1](https://doi.org/10.1016/S0957-4174(03)00007-1)
- Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245), 255–260. <https://doi.org/10.1126/science.aaa8415>
- Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2017). LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access*, 6, 1662–1669. <https://doi.org/10.1109/ACCESS.2017.2779939>
- Kayacik, H. G., & Zincir-Heywood, A. N. (2008). Mimicry Attacks Demystified: What Can Attackers Do to Evade Detection? *2008 Sixth Annual Conference on Privacy, Security and Trust*, 213–223. <https://doi.org/10.1109/PST.2008.25>
- Kephart, J. O., & Arnold, W. C. (1994). Automatic Extraction of Computer Virus Signatures. *Virus Bulletin Ltd*.
- Kiladze, T. (2019). *LifeLabs pays ransom after massive data breach affecting up to 15 million Canadians - The Globe and Mail*. The Globe and Mail. <https://www.theglobeandmail.com/business/article-lifelabs-pays-ransom-after-massive-data-breach-affecting-up-to-1/>

- Kim, G. H., & Spafford, E. H. (1994). The design and implementation of Tripwire: A file system integrity checker. *Proceedings of the ACM Conference on Computer and Communications Security*, 18–29. <https://doi.org/10.1145/191177.191183>
- Kim, J., & Kim, H. (2016). Applying Recurrent Neural Network to Intrusion Detection with Hessian Free Optimization. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (pp. 357–369). https://doi.org/10.1007/978-3-319-31875-2_30
- Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 – Conference Track Proceedings*.
- Koc, L., Mazzuchi, T. A., & Sarkani, S. (2012). A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier. *Expert Systems with Applications*. <https://doi.org/10.1016/j.eswa.2012.07.009>
- Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016). Deep Learning for Classification of Malware System Call Sequences. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 9992 LNAI* (pp. 137–149). https://doi.org/10.1007/978-3-319-50127-7_11
- Korba, J. (2000). *Windows NT Attacks for the Evaluation of Intrusion Detection Systems** *Windows NT Attacks for the Evaluation of Intrusion Detection Systems*.
- Kovari, A., & Dukan, P. (2012). KVM & OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE. *2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics, SISY 2012*, 335–339. <https://doi.org/10.1109/SISY.2012.6339540>
- Kozushko, H. (2003). Intrusion detection: Host-based and network-based intrusion detection systems. *Independent Study*.
- Krašto apsaugos ministerija. (2020). *2019-uji Nacionalinio kibernetinio saugumo vertinimas*. https://www.nksc.lt/naujienos/nacionalinio_kibernetinio_saugumo_vertinimas.html
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*.
- Kruegel, C., Mutz, D., Valeur, F., & Vigna, G. (2003). On the Detection of Anomalous System Call Arguments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (pp. 326–343). https://doi.org/10.1007/978-3-540-39650-5_19
- Lazarevic, A., Kumar, V., & Srivastava, J. (2005). Intrusion detection: A survey. In *Managing Cyber Threats* (pp. 19–78). Springer.
- Le, T. T. H., Kim, Y., & Kim, H. (2019). Network intrusion detection based on novel feature selection model and various recurrent neural networks. *Applied Sciences (Switzerland)*, 9(7). <https://doi.org/10.3390/app9071392>

- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- Lee, S. C., & Heinbuch, D. V. (2001). Training a neural-network based intrusion detector to recognize novel attacks. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans.*, 31(4), 294–299. <https://doi.org/10.1109/3468.935046>
- Lee, W., Stolfo, S. J., & Mok, K. W. (1998). Mining Audit Data to Build Intrusion Detection Models. *Stat.*, 18(5), 66–72. <http://www.aaai.org/Papers/KDD/1998/KDD98-010.pdf>
- Lhotsky, B. (2013). *Instant OSSEC Host-based Intrusion Detection*.
- Lin, P. C., Lin, Y. D., & Lai, Y. C. (2011). A hybrid algorithm of Backward Hashing and automaton tracking for virus scanning. *IEEE Transactions on Computers*. <https://doi.org/10.1109/TC.2010.95>
- Lin, Y., Zhang, Y., & Ou, Y. (2010). The Design and Implementation of Host-Based Intrusion Detection System. *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, 595–598. <https://doi.org/10.1109/IITSI.2010.127>
- Lippmann, R.P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., & Zissman, M. a. (1999). Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation. *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, 12–26 vol. 2. <https://doi.org/10.1109/DISCEX.2000.821506>
- Lippmann, Richard P., & Cunningham, R. K. (2000). Improving intrusion detection performance using keyword selection and neural networks. *Computer Networks*, 34(4), 597–603. [https://doi.org/10.1016/S1389-1286\(00\)00140-7](https://doi.org/10.1016/S1389-1286(00)00140-7)
- Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2015). Learning to Diagnose with LSTM Recurrent Neural Networks. *4th International Conference on Learning Representations, ICLR 2016 – Conference Track Proceedings*. <http://arxiv.org/abs/1511.03677>
- Liu, G., Yi, Z., & Yang, S. (2007). A hierarchical intrusion detection model based on the PCA neural networks. *Neurocomputing*, 70(7–9), 1561–1568. <https://doi.org/10.1016/j.neucom.2006.10.146>
- Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences (Switzerland)*, 9(20). <https://doi.org/10.3390/app9204396>

- Liu, M., Xue, Z., Xu, X., Zhong, C., & Chen, J. (2019). Host-Based Intrusion Detection System with System Calls. *ACM Computing Surveys*, 51(5), 1–36. <https://doi.org/10.1145/3214304>
- Liu, Z., Japkowicz, N., Wang, R., Cai, Y., Tang, D., & Cai, X. (2020). A statistical pattern based feature extraction method on system call traces for anomaly detection. *Information and Software Technology*, 126(August 2019), 106348. <https://doi.org/10.1016/j.infsof.2020.106348>
- Louridas, P., & Ebert, C. (2016). Machine Learning. *IEEE Software*, 33(5), 110–115. <https://doi.org/10.1109/MS.2016.114>
- Lyon, G. (2018). *Nmap: the network mapper*. Nmap.Org. <https://doi.org/10.1007/s00227-006-0247-y>
- Maulana, R. J., & Kusuma, G. P. (2020). Malware Classification Based on System Call Sequences Using Deep Learning. *Advances in Science, Technology and Engineering Systems Journal*, 5(4), 207–216. <https://doi.org/10.25046/aj050426>
- McGraw, G., & Morrisett, G. (2000). Attacking Malicious Code: A Report to the Infosec Research Council. *IEEE Software*, 17(5), 33–41. <https://doi.org/10.1109/52.877857>
- Mehnaz, S., & Bertino, E. (2020). A Fine-grained Approach for Anomaly Detection in File System Accesses with Enhanced Temporal User Profiles. *IEEE Transactions on Dependable and Secure Computing*, 5971(c), 1–1. <https://doi.org/10.1109/TDSC.2019.2954507>
- Meiners, C. R., Patel, J., Norige, E., Torng, E., & Liu, A. X. (2010). Fast regular expression matching using small tcams for network intrusion detection and prevention systems. *Proceedings of the 19th USENIX Security Symposium*.
- Meng, F., Fu, Y., Lou, F., & Chen, Z. (2017). An Effective Network Attack Detection Method Based on Kernel PCA and LSTM-RNN. *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*, 568–572. <https://doi.org/10.1109/ICCSEC.2017.8447022>
- Miao, W., Cheng, Z., & Jingjing, Y. (2006). Native API based windows anomaly intrusion detection method using SVM. *Proceedings – IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006 II*, 514–519. <https://doi.org/10.1109/SUTC.2006.1636219>
- Miller, C., Glendowne, D., Cook, H., Thomas, D., Lanclos, C., & Pape, P. (2017). Insights gained from constructing a large scale dynamic analysis platform. *Digital Investigation*, 22, 48–56. <https://doi.org/10.1016/j.diin.2017.06.007>
- Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., & Rajarajan, M. (2013). A survey of intrusion detection techniques in Cloud. In *Journal of Network and Computer Applications*. <https://doi.org/10.1016/j.jnca.2012.05.003>
- Monika, Zavorsky, P., & Lindskog, D. (2016). Experimental Analysis of Ransomware on Windows and Android Platforms: Evolution and Characterization. *Procedia Computer Science*. <https://doi.org/10.1016/j.procs.2016.08.072>

- Mora-Gimeno, F. J., Mora-Mora, H., Volckaert, B., & Atrey, A. (2021). Intrusion Detection System Based on Integrated System Calls Graph and Neural Networks. *IEEE Access*, 9. <https://doi.org/10.1109/ACCESS.2021.3049249>
- Moskovitch, R., Pluderman, S., Gus, I., Stopel, D., Feher, C., Parmet, Y., Shahar, Y., & Elovici, Y. (2007). Host Based Intrusion Detection using Machine Learning. *2007 IEEE Intelligence and Security Informatics*, 107–114. <https://doi.org/10.1109/ISI.2007.379542>
- Murtaza, S. S., Khreich, W., Hamou-Lhadj, A., & Couture, M. (2013). A host-based anomaly detection approach by representing system calls as states of kernel modules. *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 431–440. <https://doi.org/10.1109/ISSRE.2013.6698896>
- Nguyen, T. H., Luo, J., & Njogu, H. W. (2014). An efficient approach to reduce alerts generated by multiple IDS products. *International Journal of Network Management*. <https://doi.org/10.1002/nem.1857>
- Nisa, M., Shah, J. H., Kanwal, S., Raza, M., Khan, M. A., Damaševičius, R., & Blažauskas, T. (2020). Hybrid Malware Classification Method Using Segmentation-Based Fractal Texture Analysis and Deep Convolution Neural Network Features. *Applied Sciences*, 10(14), 4966. <https://doi.org/10.3390/app10144966>
- Norouzi, M., Souri, A., & Samad Zamini, M. (2016). A Data Mining Classification Approach for Behavioral Malware Detection. *Journal of Computer Networks and Communications*, 2016, 1–9. <https://doi.org/10.1155/2016/8069672>
- Pailoor, S., Wang, X., Shacham, H., & Dillig, I. (2020). Automated policy synthesis for system call sandboxing. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA), 1–26. <https://doi.org/10.1145/3428203>
- Paper, C. (2016). *Application of Convolutional neural networks to intrusion type recognition*. November 2015.
- Patcha, A., & Park, J. M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12), 3448–3470. <https://doi.org/10.1016/j.comnet.2007.02.001>
- Pathan, A. S. K. (2014). *The State of the Art in Intrusion Prevention and Detection*.
- Patil, S., Kashyap, A., Sivathanu, G., & Zadok, E. (2004). I3FS : An In-Kernel Integrity Checker and Intrusion Detection File System. *Proceeding LISA '04 Proceedings of the 18th USENIX Conference on System Administration*.
- Paulauskas, N., & Bagdonas, A. F. (2015). Local outlier factor use for the network flow anomaly detection. *Security and Communication Networks*, 8(18), 4203–4212. <https://doi.org/10.1002/sec.1335>
- Paulauskas, N., & Baskys, A. (2019). Application of histogram-based outlier scores to detect computer network anomalies. *Electronics (Switzerland)*, 8(11), 1–8. <https://doi.org/10.3390/electronics8111251>
- Peddoju, S. K., Upadhyay, H., Soni, J., & Prabakar, N. (2020). Natural language processing based anomalous system call sequences detection with virtual memory

- introspection. *International Journal of Advanced Computer Science and Applications*, 11(5), 455–460. <https://doi.org/10.14569/IJACSA.2020.0110559>
- Pennington, A. G., Strunk, J. D., Griffin, J. L., Soules, C. a N., Goodson, G. R., & Ganger, G. R. (2003). Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior. *Proceedings of the 12th USENIX Security Symposium*, 13(4), 137–152. <https://doi.org/10.1145/1880022.1880024>.http
- Radiuk, P. M. (2018). Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. *Information Technology and Management Science*, 20(1), 20–24. <https://doi.org/10.1515/itms-2017-0003>
- Radziwill, N. M. (2018). Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon. *Quality Management Journal*. <https://doi.org/10.1080/10686967.2018.1436358>
- Raut, A. S., & Singh, K. R. (2014). Anomaly Based Intrusion Detection-A Review. *Int. J. on Network Security*, 5. <https://doi.org/01.IJNS.4.206.0526>
- Reuning, J. R. (2004). *Applying term weight techniques to event log analysis for intrusion detection*.
- Rhodes, B. C., Mahaffey, J. A., & Cannady, J. D. (2000). Multiple self-organizing maps for intrusion detection. *Proceedings of the 23rd ...*, 16–19. <https://doi.org/10.1.1.10.9322>
- Rice, R. E., & Borgman, C. L. (1983). The use of computer-monitored data in information science and communication research. *Journal of the American Society for Information Science*, 34(4), 247–256.
- Roesch, M. (1999). *Snort – Lightweight Intrusion Detection for Networks*. 99, 229–238. <http://www.usenix.org>
- Ronen, R., & Feuerstein, C. (2015). *Microsoft Malware Classification Challenge (BIG 2015) | Kaggle*. <https://www.kaggle.com/c/malware-classification/overview>
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*. <https://doi.org/10.1007/s11263-015-0816-y>
- Ryan, J., Lin, M. J., & Mikkulainen, R. (1998). Intrusion detection with neural networks. *Advances in Neural Information ...*. <http://www.aaai.org/Papers/Workshops/1997/WS-97-07/WS97-07-013>
- Sadoddin, R., & Ghorbani, A. (2006). Alert correlation survey. *Proceedings of the 2006 International Conference on Privacy, Security and Trust Bridge the Gap Between PST Technologies and Business Services – PST '06*, 1. <https://doi.org/10.1145/1501434.1501479>
- Sahu, S. K., Sarangi, S., & Jena, S. K. (2014). A detail analysis on intrusion detection datasets. *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014*, 1348–1353. <https://doi.org/10.1109/IAAdCC.2014.6779523>

- Sanz, J. A., Bernardo, D., Herrera, F., Bustince, H., & Hagsras, H. (2015). A Compact Evolutionary Interval-Valued Fuzzy Rule-Based Classification System for the Modeling and Prediction of Real-World Financial Applications with Imbalanced Data. *IEEE Transactions on Fuzzy Systems*. <https://doi.org/10.1109/TFUZZ.2014.2336263>
- Sazzadul Hoque, M. (2012). An Implementation of Intrusion Detection System Using Genetic Algorithm. *International Journal of Network Security & Its Applications*. <https://doi.org/10.5121/ijnsa.2012.4208>
- Scarfone, K., Mell, P., & Mell, P. (2007). *NIST Special Publication 800-94 IDS*.
- Schwartzbard, A. K. G. and A. (1999). A study of using neural network for anomaly and misuse detection. *Proceedings of the 8th USENIX Security Symposium*. <http://dblp.uni-trier.de/db/conf/uss/uss1999.html#GhoshS99>
- Sekeh, M. A., & Maarof, M. A. Bin. (2009). Fuzzy Intrusion Detection System via Data Mining Technique with Sequences of System Calls. *2009 Fifth International Conference on Information Assurance and Security*, 1(November), 154–157. <https://doi.org/10.1109/IAS.2009.32>
- Shah, S. A. R., & Issac, B. (2018). Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Generation Computer Systems*, 80, 157–170. <https://doi.org/10.1016/j.future.2017.10.016>
- Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy, 2018-Janua(Cic)*, 108–116. <https://doi.org/10.5220/0006639801080116>
- Shyu, M.-L., Chen, S.-C., Sarinnapakorn, K., & Chang, L. (2003). A Novel Anomaly Detection Scheme Based on Principal Component Classifier. *Dtic, ADA465712*. <https://doi.org/10.1007/11539827-18>
- Singaravelan, S., Arun, R., Arunshunmugam, D., Joy, S. J. C., & Murugan, D. (2020). Inner interruption discovery and defense system by using data mining. *Journal of King Saud University – Computer and Information Sciences*, 32(5), 592–598. <https://doi.org/10.1016/j.jksuci.2017.09.009>
- Smaha, S. E. (1988). Haystack: an intrusion detection system. In *[Proceedings 1988] Fourth Aerospace Computer Security Applications* (pp. 37–44). <https://doi.org/10.1109/ACSAC.1988.113412>
- Smith, R. G., & Eckroth, J. (2017). Building AI Applications: Yesterday, Today, and Tomorrow. *AI Magazine*, 38(1), 6. <https://doi.org/10.1609/aimag.v38i1.2709>
- Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C.-L., Levitt, K. N., Mukherjee, B., Smaha, S. E., Grance, T., Teal, D. M., & Mansur, D. (1991). DIDS (Distributed intrusion detection system) – Motivation, architecture, and an early prototype. *Proceedings of the 14th National Computer Security Conference*. <https://doi.org/citeulike-article-id:9827035>

- Srivastav, A., Tiwari, A., & Singh, K. V. (2020). Host-based Intrusion Detection System (HIDS). *International Journal of Engineering and Advanced Technology*, 9(5), 1043–1049. <https://doi.org/10.35940/ijeat.E9903.069520>
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*.
- Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A multi-class classification competition. *Proceedings of the International Joint Conference on Neural Networks*, 1453–1460. <https://doi.org/10.1109/IJCNN.2011.6033395>
- StatCounter Global Stats. (2019). *Desktop Operating System Market Share Worldwide*. <https://gs.statcounter.com/os-market-share/desktop/worldwide/2019>
- Staudemeyer, R. C. (2015). Applying long short-term memory recurrent neural networks to intrusion detection. *South African Computer Journal*, 56(1). <https://doi.org/10.18489/sacj.v56i1.248>
- Stolfo, S. J., Apap, F., Eskin, E., Heller, K., Hershkop, S., Honig, A., & Svore, K. (2005). A comparative evaluation of two algorithms for Windows Registry Anomaly Detection. *Journal of Computer Security*. <https://doi.org/10.3233/JCS-2005-13403>
- Sun, J., Yang, H., Tian, J., & Wu, F. (2009). Intrusion Detection Method Based on Wavelet Neural Network. *2009 Second International Workshop on Knowledge Discovery and Data Mining*, 851–854. <https://doi.org/10.1109/WKDD.2009.214>
- Sung, A. H., & Mukkamala, S. (2003). Identifying Important Features for Intrusion Detection Using Support Vector Machines and Neural Networks. *Proceedings of the Symposium on Applications and the Internet (SAINT'03)*, 1(1), 209–216. <https://doi.org/10.1109/SAINT.2003.1183050>
- Symantec. (2019). *Internet security threat report 2019*. <https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf>
- Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. <https://doi.org/10.1109/CVPR.2015.7298594>
- Szor, P. (2005). *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional.
- Tan, K. M. C., & Maxion, R. A. (2002). “Why 6?” Defining the operational limits of stide, an anomaly-based intrusion detector. *Proceedings - IEEE Symposium on Security and Privacy*. <https://doi.org/10.1109/SECPRI.2002.1004371>
- Tandon Gaurav, & Philip K. Chan. (2005). Learning Useful System Call Attributes for Anomaly Detection. *FLAIRS Conference*, 405–411. <http://dblp.uni-trier.de/db/conf/flairs/flairs2005.html#TandonC05>

- Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2018). Deep Recurrent Neural Network for Intrusion Detection in SDN-based Networks. *2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018, NetSoft*, 462–469. <https://doi.org/10.1109/NETSOFT.2018.8460090>
- Thales. (2019). *2019 Thales Data Threat Report – Global Edition*. <https://www.thalesecurity.com/2019/data-threat-report>
- thezoo.morirt.com. (n.d.). *theZoo aka Malware DB by ytisf*. Retrieved December 27, 2017, from <http://thezoo.morirt.com/>
- Timofte, J. (2008). Intrusion Detection using Open Source Tools. *Architecture*, 2(2), 75–79. <http://www.revistaie.ase.ro/content/46/Timofte.pdf>
- Tirumala, S. S., Sathu, H., & Sarrafzadeh, A. (2015). Free and open source intrusion detection systems: A study. *Proceedings - International Conference on Machine Learning and Cybernetics*, 1, 205–210. <https://doi.org/10.1109/ICMLC.2015.7340923>
- Topallar, M., Ozgur Depren, M., Anarim, E., & Ciliz, K. (n.d.). Host-based intrusion detection by monitoring windows registry accesses. *Proceedings of the IEEE 12th Signal Processing and Communications Applications Conference*, 2004, 728–731. <https://doi.org/10.1109/SIU.2004.1338634>
- Tran, N. N., Sarker, R., & Hu, J. (2018). An Approach for Host-Based Intrusion Detection System Design Using Convolutional Neural Network. In J. Hu, I. Khalil, Z. Tari, & S. Wen (Eds.), *Mobile Networks and Management* (pp. 116–126). Springer International Publishing. https://doi.org/10.1007/978-3-319-90775-8_10
- Ullah, W., Ullah, A., Haq, I. U., Muhammad, K., Sajjad, M., & Baik, S. W. (2020). CNN features with bi-directional LSTM for real-time anomaly detection in surveillance networks. *Multimedia Tools and Applications*. <https://doi.org/10.1007/s11042-020-09406-3>
- Vaas, C., & Happa, J. (2017). Detecting disguised processes using application-behavior profiling. *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, 1–6. <https://doi.org/10.1109/THS.2017.7943508>
- Venckauskas, A., Jusas, V., Paulikas, K., & Toldinas, J. (2017). Methodology to investigate BitTorrent sync protocol. *Computer Science and Information Systems*, 14(1), 197–218. <https://doi.org/10.2298/CSIS160212032V>
- Vigna, G., & Kemmerer, R. A. (1998). NetSTAT: A network-based intrusion detection approach. *Proceedings - Annual Computer Security Applications Conference, ACSAC*. <https://doi.org/10.1109/CSAC.1998.738566>
- Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2017). Applying convolutional neural network for network intrusion detection. *2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017, 2017-Janua*, 1222–1228. <https://doi.org/10.1109/ICACCI.2017.8126009>
- VirusShare.com. (2020). *VirusShare.com*. <https://virusshare.com/>
- VirusTotal. (2020). *VirusTotal*. <https://www.virustotal.com>

- Walker, D. (2014). "Havex" malware strikes industrial sector via watering hole attacks. SC Magazine. <https://www.scmagazine.com/home/security-news/havex-malware-strikes-industrial-sector-via-watering-hole-attacks/>
- Wang, G., Hao, J., Ma, J., Huang, L., Mab, J., & Huang, L. (2010). A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering. *Expert Systems with Applications*, 37(9), 6225–6232. <https://doi.org/10.1016/j.eswa.2010.02.102>
- Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., & Xu, W. (2016). CNN-RNN: A Unified Framework for Multi-label Image Classification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-Decem*, 2285–2294. <https://doi.org/10.1109/CVPR.2016.251>
- Wang, Zhiguang, Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. *2017 International Joint Conference on Neural Networks (IJCNN)*, 1578–1585. <https://doi.org/10.1109/IJCNN.2017.7966039>
- Wang, Zhijian, & Zhu, Y. (2017). A centralized HIDS framework for private cloud. *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 115–120. <https://doi.org/10.1109/SNPD.2017.8022709>
- Wei, L., Xiao, A., Xie, L., Chen, X., Zhang, X., & Tian, Q. (2020). *Circumventing Outliers of AutoAugment with Knowledge Distillation*. <http://arxiv.org/abs/2003.11342>
- Wen, G., Hou, Z., Li, H., Li, D., Jiang, L., & Xun, E. (2017). Ensemble of Deep Neural Networks with Probability-Based Fusion for Facial Expression Recognition. *Cognitive Computation*, 9(5), 597–610. <https://doi.org/10.1007/s12559-017-9472-6>
- Witten, I. H., Frank, E., & Hall, M. A. (2005). Data Mining: Practical machine learning tools and techniques. In *Data Mining*. <https://doi.org/0120884070>, 9780120884070
- Wu, X., Kumar, V., Ross, Q. J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z. H., Steinbach, M., Hand, D. J., & Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1), 1–37. <https://doi.org/10.1007/s10115-007-0114-2>
- Wunderlich, S., Ring, M., Landes, D., & Hotho, A. (2020). The Impact of Different System Call Representations on Intrusion Detection. *Logic Journal of the IGPL*, 00(0). <https://doi.org/10.1093/jigpal/jzaa058>
- Xie, M., & Hu, J. (2013). Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD. *Proceedings of the 2013 6th International Congress on Image and Signal Processing, CISP 2013*, 3(Cisp), 1711–1716. <https://doi.org/10.1109/CISP.2013.6743952>
- Xie, M., Hu, J., & Slay, J. (2014). Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD. *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 978–982. <https://doi.org/10.1109/FSKD.2014.6980972>

- Xie, M., Hu, J., Yu, X., & Chang, E. (2014). Evaluating Host-Based Anomaly Detection Systems: Application of the Frequency-Based Algorithms to ADFA-LD. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (pp. 542–549). https://doi.org/10.1007/978-3-319-11698-3_44
- Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., & Wang, C. (2018). Machine Learning and Deep Learning Methods for Cybersecurity. *IEEE Access*, 6, 35365–35381. <https://doi.org/10.1109/ACCESS.2018.2836950>
- 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, 48697–48707. <https://doi.org/10.1109/ACCESS.2018.2867564>
- Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4), 611–629. <https://doi.org/10.1007/s13244-018-0639-9>
- Yang, Y., McLaughlin, K., Sezer, S., Littler, T., Im, E. G., Pranggono, B., & Wang, H. F. (2014). Multiattribute SCADA-specific intrusion detection system for power networks. *IEEE Transactions on Power Delivery*. <https://doi.org/10.1109/TPWRD.2014.2300099>
- Ye, Q., Wu, X., & Yan, B. (2010). An Intrusion Detection Approach Based on System Call Sequences and Rules Extraction. *2010 2nd International Conference on E-Business and Information System Security*, 1–4. <https://doi.org/10.1109/EBISS.2010.5473675>
- Yong, H., & Feng, Z. X. (2010). Expert System Based Intrusion Detection System. *2010 3rd International Conference on Information Management, Innovation Management and Industrial Engineering*, 404–407. <https://doi.org/10.1109/ICIII.2010.578>
- Yu, Y., Ge, Y., & Fu-xiang, G. (2005). A neural network approach for misuse and anomaly intrusion detection. *Wuhan University Journal of Natural Sciences*, 10(1), 115–118. <https://doi.org/10.1007/BF02828630>
- Yuanyuan, C., & Zhibin, W. (2018). Quantitative analysis modeling of infrared spectroscopy based on ensemble convolutional neural networks. *Chemometrics and Intelligent Laboratory Systems*, 181(March), 1–10. <https://doi.org/10.1016/j.chemolab.2018.08.001>
- Zhang, S., Xie, X., & Xu, Y. (2019). Intrusion detection method based on a deep convolutional neural network. *Qinghua Daxue Xuebao/Journal of Tsinghua University*. <https://doi.org/10.16511/j.cnki.qhdxxb.2019.22.004>
- Zhang, Z., Li, J., Manikopoulos, C. N., Jorgenson, J., & Ucles, J. (2001). HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. *Proceedings of the IEEE Workshop on Information Assurance and Security*, 85–90.

List of Scientific Publications by the Author on the Topic of the Dissertation

Papers in the Reviewed Scientific Journals

Čeponis, D., & Goranin, N. (2020). Investigation of Dual-Flow Deep Learning Models LSTM-FCN and GRU-FCN Efficiency against Single-Flow CNN Models for the Host-Based Intrusion and Malware Detection Task on Univariate Times Series Data. *Applied Sciences*, 10(7), 2373. <https://doi.org/10.3390/app10072373>. [Science Citation Index Expanded (Web of Science)] [Citav. rod.: 2,474 (2019, InCites JCR SCIE)]

Čeponis, D., & Goranin, N. (2019). Evaluation of Deep Learning Methods Efficiency for Malicious and Benign System Calls Classification on the AWSCTD. *Security and Communication Networks*, 2019, 1–12. <https://doi.org/10.1155/2019/2317976>. [Science Citation Index Expanded (Web of Science)] [Citav. rod.: 1,288 (2019, InCites JCR SCIE)]

Čeponis, D., & Goranin, N. (2018). Towards a Robust Method of Dataset Generation of Malicious Activity for Anomaly-Based HIDS Training and Presentation of AWSCTD Dataset. *Baltic Journal of Modern Computing*, 6(3). <https://doi.org/10.22364/bjmc.2018.6.3.01>

Goranin, N., & Čeponis, D. (2018). Investigation of AWSCTD dataset applicability for malware type classification. *International Scientific Journal Security & Future*, 2(4), 186–189.

Radvilavičius, L., & Čeponis, D. (2011). Window API funkcijų sekų perėmimo bibliotekų tyrimas. *Mokslas – Lietuvos Ateitis*, 3(1), 15–19. <https://doi.org/10.3846/mla.2011.003>.

Papers in Other Editions

Čeponis, D., & Goranin, N. (2018). Towards a robust method of dataset generation of malicious activity on a windows-based operating system for anomaly-based HIDS training. *CEUR Workshop Proceedings*, 2158.

Goranin, N., & Čeponis, D. (2018). Investigation of AWSCTD dataset applicability for malware type classification. *II International Scientific Conference CONFSEC, 10–13 December 2018 Borovets, Bulgaria. Vol. 2. Information Security, Technical Facilities for Ensuring Security*, 2(2), 152–155.

Čeponis, D., & Radvilavičius, L. (2008). Windows API funkcijų stebėjimas, procesų perėmimas ir stebėjimas. *11-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“ 2008 metų teminės konferencijos INFORMATIKA (2008 m. balandžio 9–11 d.) straipsnių rinkinys*, 391–396.

Summary in Lithuanian

Įvadas

Problemos formulavimas

Dėl nuolatinio kibernetinių atakų daugėjimo būtina naudoti vis sudėtingesnius kompiuterio lygmens kenksmingo kodo ir įsibrovimo į sistemas aptikimo metodus. Plačiai taikomas parašais grindžiamas įsibrovimo aptikimo metodas nepajėgia aptikti nulinės dienos ir būsimų atakų. Be to, atakos, pritaikytos tam tikroms organizacijoms, taip pat lieka nepastebėtos, jei taikomi parašais grindžiami metodai (Xie & Hu, 2013).

Kita problema ta, kad įsibrovimo aptikimas, paremtas tinklo duomenimis, yra plačiau naudojamas ir turi pakankamą kiekį tam pritaikytų duomenų rinkinių. Priešingai, kompiuterio lygmens kenksmingo programinio kodo ir įsibrovimo aptikimo sistemoms trūksta kokybiškų laisvai prieinamų duomenų rinkinių. Tačiau HIDS ir AV sistemos pastaruoju metu sulaukia vis daugiau dėmesio, nes sugeba pateikti daugiau informacijos apie patį įsibrovimą, gali apsaugoti nuo reikšmingos žalos (pvz., kritinių sistemos failų modifikavimo) bei suteikia papildomą apsaugos sluoksnį (Kozushko, 2003). Taip pat dauguma esamų HIDS duomenų rinkinių yra orientuoti į „Linux“ operacinę sistemą ir turi mažai „Windows“ operacinės sistemos įrašų, kuri pagal statistinius duomenis yra populiariausia tarp stalinių kompiuterių (StatCounter Global Stats, 2019).

Anomalijomis grindžiami metodai padeda aptikti daugiau kibernetinių atakų, ypač nulinės dienos ir polimorfinio kenksmingo kodo atvejus. Pagrindinė šių metodų mintis – susidaryti standartinį (nekenksmingą) vartotojo arba sistemos elgsenos modelį ir įvykius,

neatitinkančius to modelio, pažymėti kaip įsibrovimą (naudojant tam tikras ribines vertes) (Aldweesh et al., 2020).

Pastaraisiais metais, padidėjus kompiuterinės įrangos spartai (ypač kompiuterių vaizdo plokščių), giliojo apmokymo modeliai padėjo padidinti anomalijomis pagrįstų įsibrovimo aptikimų sistemų tikslumą ir sumažino klaidingų aptikimų dažnumą (Hwang et al., 2019).

Darbo aktualumas

Mokslininkai taiko sudėtingus mašininio apmokymo metodus, kad pasiektų realiame pasaulyje naudotinas kenksmingo kodo ir įsibrovimo aptikimo sistemų charakteristikas. Pasirinkti būdai turi kelias problemas: jiems apmokyti reikalingi dideli kiekiai duomenų ir reikalinga sparti aparatinė įranga.

Tinklo duomenų gausa taip pat verčia mokslininkus domėtis tik vienu problemos aspektu. Aptikus kenksmingus veiksmus viename informacinės sistemos sluoksnyje, galima tų pačių atakų neaptikti kitame sluoksnyje, pvz., kompiuterio lygmenyje. Didžioji dalis duomenų rinkinių sufleruoja, kad įsibrovimai įvyksta tinklo lygmeniu ir kompiuteriuose su „Linux“ operacine sistema. Bet remiantis statistiniais duomenimis, „Windows“ operacinė sistema vis dar yra populiariausia tarp galutinių vartotojų.

Duomenų rinkinių problema gali būti išspręsta paprastai – turi būti pasiūlytas ir įgyvendintas naujas patikimas trūkstamų duomenų surinkimo metodas. Naujas duomenų rinkinys bus tolimesnių tyrimų, susijusių su ML ir DL metodų efektyvumu, juos taikant kompiuterio lygmens kenksmingam kodui ir įsibrovimui aptikti, pagrindas.

Atliktas skirtingų ML ir DL metodų efektyvumo tyrimas, naudojant naują AWSCTD duomenų rinkinį, leido nustatyti pačius tinkamiausius iš jų. Ateityje jie galės būti praktiškai panaudoti kenksmingo kodo ir įsibrovimo sistemose ir tolesniems tyrimams.

Tyrimo objektas

Pagrindinis šio tyrimo objektas yra ML ir DL metodų pritaikymas kompiuterio lygmens kenksmingam programiniam kodui ir įsibrovimui aptikti informacinėse sistemose.

Darbo tikslas

Disertacijos tikslas – pagerinti anomalijomis pagrįstų kompiuterio lygmens kenksmingo programinio kodo ir įsibrovimų aptikimo metodų įvėrčius, taikant ML ir DL metodus.

Darbo uždaviniai

Nustatyto darbo tikslui pasiekti būtina išspręsti toliau pateiktus darbo tikslus:

1. Atlikti mokslinės literatūros apžvalgą apie dabar kenksmingam programiniam kodui ir įsibrovimui aptikti taikomus ML ir DL metodus bei duomenų rinkinius, skirtus „Windows“ operacinėms sistemoms.

2. Pasiūlyti patikimą metodą, skirtą išsamaus kenksmingo ir nekenksmingo programinio kodo elgsenos duomenų rinkiniui generuoti, kuris vėliau galėtų būti panaudotas ML ir DL metodų efektyvumui tirti, juos pritaikius kompiuterio lygmens įsibrovimui aptikti informacinėse sistemose.
3. Atlikti kompiuterio lygmens įsibrovimo ir kenksmingo programinio kodo aptikimo uždavinių eksperimentinį vertinimą ir pritaikymo analizę, naudojant nustatytus įverčius ir šiuos metodus:
 - 3.1. Plačiausiai naudojamus ML metodus;
 - 3.2. Plačiausiai naudojamus paprastus DL metodus;
 - 3.3. Esamus ir naujai pasiūlytus / modifikuotus sudėtingus DL metodus.

Tyrimų metodika

Šiame darbe buvo naudojama kiekybinė tyrimų metodika. Siekiant geriau suprasti įsibrovimo ir kenkėjiškų programų aptikimo galimybes, buvo atlikta dabartinių metodų ir naudojamų duomenų rinkinių apžvalga, daugiausia dėmesio skiriant pagrindinio kompiuterio lygmens atakoms. Vėliau buvo atlikti išsamūs eksperimentai, siekiant surinkti naują duomenų rinkinį įsibrovimui ir kenksmingam programiniam kodui aptikti bei nustatyti tam tinkamiausius ML ir DL metodus. Naujam duomenų rinkiniui surinkti buvo naudojamas „HP ProLiant DL 380 G6“ serveris ir autoriaus pasiūlytas metodas. Geriausiems kenksmingo programinio kodo klasifikavimo ML metodams nustatyti buvo naudojamas kompiuteris su „Intel i7-7500U“ 2,70 GHz procesoriumi, 8 GB RAM ir „Windows 10“ operacine sistema bei „Weka 3.8“ programine įranga. Geriausiems DL įsibrovimo ir kenksmingo programinio kodo aptikimo metodams nustatyti buvo naudojamas kompiuteris su „Intel i5-3570“ 3,80 GHz procesoriumi, GTX 1070 vaizdo plokšte, 16 GB RAM, „Ubuntu 18.04.3 LTS“ operacine sistema ir „Keras“ bei „Tensorflow“ bibliotekomis. Surinkti duomenys buvo lyginami naudojant standartinius ML ir DL įverčius: tikslumas, preciziškumas, atšaukimo statistika, f-1 statistikos reikšmė, klaidingai teigiamas rodiklis, klaidingai neigiamas rodiklis ir klasifikavimo klaida.

Darbo mokslinis naujumas

Darbo mokslinis naujumas pagrįstas šiais rezultatais:

1. Sukurtas naujas kompiuterio lygmens duomenų rinkinys (AWSCTD), kuris yra tinkamas anomalijomis pagrįstam įsibrovimų ir kenksmingo kodo aptikimui „Windows“ operacinėje sistemoje. Jame sukauptas platus kenksmingo ir nekenksmingo kodo elgsenos duomenų kiekis, kuris yra išsamesnis bei įvairesnis, nei šiuo metu esami duomenų rinkiniai. Duomenų rinkinys buvo sudarytas naudojant darbo autoriaus pasiūlytą patikimą metodą.
2. Pasiūlytas naujas paprasto konvoliucinių neuroninių tinklų modelio pritaikymas kenksmingoms ir nekenksmingoms aplikacijoms klasifikuoti naudojant sisteminių funkcijų kvietimo sekas kaip duomenų vektorių. Eksperimentiškai nustatytas modelis beveik visose vertinimo metrikose parodė

panašius arba geresnius rezultatus nei kitų mokslininkų naudojami pažangesni konvoliucinių arba rekurentinių neuroninių tinklų pagrindu sukurti modeliai, kartu išsaugodamas geresnius laiko sąnaudų rezultatus. Modeliams apmokyti naudojant AWSCTD duomenų rinkinį, buvo pasiektas 99,1–99,3 % tikslumas, klasifikuojant kenksmingas ir nekenksmingas programas.

3. Pristatytas naujas, prieš tai nenaudotas konvoliuciniais neuroniniais tinklais grindžiamas vieno srauto modelis, tinkamas anomalijomis pagrįstam įsibrovimų ir kenksmingo kodo aptikimui naudojant programų sisteminių funkcijų sekas kaip duomenų vektorių. Pasiūlytas metodas pateikia geresnius tikslumo, apmokymo ir klasifikavimo laikų rezultatus, lyginant su pažangiausiais dviejų srautų modeliais, kai naudojamas AWSCTD duomenų rinkinys.

Darbo rezultatų praktinė reikšmė

Disertacijos pasiūlytų metodų teorinė ir praktinė reikšmė yra svarbi kritiniam anomalijomis pagrįstam įsibrovimų ir kenksmingo kodo aptikimui, nes vis daugiau naujų kibernetinių atakų yra tikslinės ir įprastiniai parašais grindžiami metodai nepajėgūs nuo jų apsaugoti.

Naujai sukurtas išsamus duomenų rinkinys AWSCTD leis mokslininkams toliau gerinti anomalijomis paremtas kompiuterio lygmens įsibrovimo sistemas. Tai svarbu, nes dabar vyrauja tinklo lygmens duomenų rinkiniai arba esantys kompiuterio lygmens rinkiniai yra neišsamūs.

Pasiektas 99,1–99,3 % klasifikavimo tikslumas ir laikas (0,00223 s), skirtas vienai kenksmingai / nekenksmingai klasifikavimo užduočiai (įsibrovimo aptikimui) atlikti, užtikrina pasiūlytų metodų ir modelių praktinį patikimumą ir greitaveiką komercinėse įsibrovimo aptikimo ir antivirusinėse sistemose kartu sujungiant parašais grindžiamus metodus, kurie užtikrintų žinomų atakų aptikimą, kai anomalijomis pagrįsti metodai (konvoliuciniais neuroniniais tinklais paremti paprastas dinaminis ir naujai pasiūlytas vieno srauto modeliai) būtų taikomi neatpažintiems įtartinėms atvejams įvertinti.

Ginamieji teiginiai

Išanalizavus tyrimo metu nustatytą informaciją ir įvertinus darbo rezultatus, buvo suformuluoti toliau pateikti ginamieji teiginiai:

1. ML metodai yra neperspektyvūs, palyginti su DL metodais, kai taikomi kompiuterio lygmens įsibrovimui ir kenksmingam programiniam kodui aptikti, naudojant kenksmingo programinio kodo sisteminių funkcijų sekas, nes pasiektas tikslumas yra palyginti mažas (90–92 %).
2. Paprasti DL modeliai yra efektyvesni už sudėtingus pažangius dviejų srautų statinius giliojo apmokymo modelius, kai naudojami vienanariai duomenys kompiuterio lygmens įsibrovimo ir kenksmingo programinio kodo aptikimo uždaviniui spręsti, naudojant AWSCTD duomenų rinkinį.

3. Pirmųjų 600 „Windows“ operacinės sistemos sisteminių funkcijų sekos naudojimas leidžia pasiekti didesnę nei 95 % aptikimo tikslumą, kuris yra pakankamas nustatytiems arba pasiūlytiems DL metodams pritaikyti hibridiniuose arba į dideles įmones orientuotuose sprendimuose, kurie sujungia automatinį didžiosios dalies anomalijų aptikimą, palikdami neaiškių atvejų analizę ekspertams, o pirmų 1000 sisteminių funkcijų naudojimas leidžia užtikrinti visišką aptikimo proceso automatizavimą ir 99,3 % tikslumą.

Darbo rezultatų aprobavimas

Disertacijos tema paskelbtos 6 mokslinės publikacijos, iš kurių 3 – žurnaluose, įtrauktuose į *Clarivate Analytics* (buv. *Thomson Reuters*) *Web of Science* duomenų bazę, 3 – mokslinių konferencijų pranešimų rinkiniuose. Moksliniai rezultatai buvo pristatyti 6 mokslinėse konferencijose:

- The 13th Junior Researchers Conference “Science – Future of Lithuania” on IT Security and Information Systems, 2010, Vilnius, Lithuania.
- The 20th Junior Researchers Conference “Science – Future of Lithuania” on IT Security and Information Systems, 2017, Vilnius, Lithuania.
- The 13th International Baltic Conference On Databases And Information Systems (Baltic DB&IS 2018), Trakai, Lithuania.
- The Second International Scientific Conference on Security (CONFSEC 2018), Borovetsas, Bulgarija.
- The 24th International Nordic Conference on Secure IT Systems (NordSec 2019), Aalborgas, Danija.
- The 11th International Workshop Data Analysis Methods for Software Systems (DAMSS 2019), Druskininkai, Lithuania.

Disertacijos struktūra

Disertaciją sudaro įvadas, keturi pagrindiniai skyriai, bendrosios išvados, literatūros šaltinių sąrašas, autoriaus publikacijų disertacijos tema sąrašas, santrauka lietuvių kalba. Darbo apimtis – 156 puslapiai neskaitant priedų, tekste yra 54 paveikslai ir 47 lentelės. Rašant disertaciją buvo panaudota 228 literatūros šaltiniai.

1. Kompiuterio lygmens įsibrovimo ir kenksmingo programinio kodo aptikimo informacinėse sistemose literatūros analizė

Šiame skyriuje apžvelgiami metodai, taikomi kibernetinėms atakoms, nukreiptoms prieš informacines sistemas, aptikti. Pateikiamas pirminis įsibrovimo aptikimo apibrėžimas kartu su skirtingais jo įgyvendinimo būdais, kurie nusakomi pagal įdiegimo vietą tinkle, vartotojo kompiuteryje arba abiejuose vietose (hibridinis būdas). Aptariami įvairūs įsib-

rovimo duomenų analizės būdai: statistiniai, mašininio ir giliojo apmokymo. Skyriaus pa-baigoje pateikiamos apžvalgos išvados ir uždavinių bei darbo tyrimo kryptių suformula-vimas.

Informacijos apsauga yra viena iš pagrindinių šiandieninių kompanijų uždavinių. Jei ši užduotis atliekama netinkamai, kompanija gali prarasti svarbius duomenis arba netgi kritinę infrastruktūrą. Remiantis Nacionalinio kibernetinio saugumo centro 2019 metų ataskaita, kibernetinių nusikaltimų skaičius per metus padidėjo tris kartus (Krašto apsaugos ministerija, 2020).

IDS naudojamos kaip vienas iš saugumo sluoksnių, norint atlikti kenksmingo prog-raminio kodo ir įsibrovimo aptikimo užduotį. NIDS ir HIDS yra vieni populiariausių būdų, turintys savų privalumų ir trūkumų. NIDS yra skirtos aptikti atakas tinklo lygmenyje, o HIDS – kompiuterio lygmenyje. NIDS sistemos analizuoja tinklo duomenis ir gali vienu metu stebėti daug kompiuterių bei nustatyti išorines grėsmes, prieš joms išsplantant tinkle. Kita vertus, NIDS turi ribotą galimybę išanalizuoti visus duomenis didelio pralaidumo tinkle (Bhuyan et al., 2015). Pagrindiniai NIDS komponentai (anomalijomis pagrįstos): tinklo duomenų perėmimo modulis, anomalijos aptikimo modulis, žinomų atakų duome-nys, konfigūracijos duomenys, apdorojimo modulis ir saugumo specialistas. Verta pami-nėti, kad pagrindinis NIDS minusas – užšifruoti tinklo duomenys. Tokiu atveju sistema negalės jų analizuoti ir įvertinti. HIDS analizuoja informaciją, pasiekiamą kompiuterio lygmenyje. Tam reikalinga papildoma programinė įranga kiekviename stebimame kom-piuteryje arba serveryje. HIDS gali pasinaudoti šiais duomenimis, kad atliktų savo už-duotį: sistemos žurnalų ir audito įrašai, sistemos funkcijų iškvietai, „Windows“ registru pakeitimai, failų sistemos pakeitimai, sistemos našumas (Grimmer et al., 2018).

Klasikinės IDS naudoja parašais grindžiamą įsibrovimo aptikimą, kuris pritaikytas žinomoms atakoms atlikti ir turi mažą klaidingų teigiamų pranešimų skaičių. Naujesnės IDS bando pritaikyti anomalijomis paremtą įsibrovimų aptikimą, kai nustatomas tipinis vartotojo elgsenos modelis ir stiprūs nukrypimai nuo jo traktuojami kaip įsibrovimas. Toks būdas yra labai efektyvus prieš nulinės dienos atakas (Chandola et al., 2009). Nors parašais grindžiamas įsibrovimo aptikimas yra gana populiarus, jis buvo sukurtas dar tada, kai kenksmingas programinis kodas plisdavo daug lėčiau ir buvo panašesnis. Šiandien, kai egzistuoja polimorfinis kenksmingas programinis kodas ir tikslinės atakos, parašais grin-džiamas aptikimas tampa nebeefektyvus (Symantec, 2019).

Anomalijomis pagrįsti kenksmingo programinio kodo ir įsibrovimo aptikimo meto-dai skirstomi į tris grupes: statistiniai, paremti žiniomis ir paremti mašininio apmokymo. ML ir DL metodų pritaikymas dabar yra bene populiariausias būdas pagerinti IDS sistemų efektyvumą. Populiariausi ML metodai, pritaikyti šiam uždaviniui, yra šie: paramos vek-torių mašinos, naivusis Bajeso tinklas, klasterių analizė, genetiniai algoritmai, paslėpti Markovo modeliai, sprendimo medžiai, neuroniniai tinklai. Nuo 2006 m. atsirado DL me-todų, kurie savo efektyvumu smarkiai aplenkė standartinius ML metodus (Deng & Yu, 2013). Išryškėjo du pagrindiniai DL modeliai – CNN ir RNN. Pagrindinis CNN pritaiky-mas – vaizdo atpažinimo sritis. Pagrindiniai šio modelio veikimo etapai: transformuoti pradinius duomenis pagal tam tikrus filtrus, sumažinti duomenų kiekį panaudojus sutel-kimo sluoksnį, pritaikyti NN kaip paskutinį klasifikavimo žingsnį. RNN modeliai yra pri-taikyti dirbti su chronologiniais duomenimis ir gali atlikti tiek klasifikavimo, tiek naujų sekų generavimo užduotis.

Norint pritaikyti ML ir DL modelius, reikalingi išsamūs duomenų rinkiniai. Ilgą laiką dominavo NIDS sistemoms pritaikytas KDD Cup 99 rinkinys, sukurtas 1999 m. (Brugger, 2007; Lippmann et al., 1999). Norint ištaisyti šią spragą, buvo pasiūlyta keletas naujų išsamių duomenų rinkinių: „FirefoxDS“ (Murtaza et al., 2013), ADFA-LD (Creech & Hu, 2013) ir NGIDS-DS (Haider et al., 2017). Bet deja, visi jie buvo orientuoti į „Linux“ operacinę sistemą. Žinoma, buvo sukurti ir „Windows“ operacinės sistemos duomenų rinkiniai, skirti kompiuterio lygmens kenksmingo programinio kodo ir įsibrovimo aptikimo tyrimams. 2016 m. Australijos mokslininkų buvo pristatytas ADFA-IDS duomenų rinkinys, apimantis „Windows“ operacinės sistemos sisteminių funkcijų kvietimo sekas (Creech & Hu, 2014; Waqas Haider et al., 2016). Jis turi dvi spragas: nepakankamas ir neišsamus duomenų kiekis. 2018 m. buvo pristatytas CSE-CIC-IDS2018 duomenų rinkinys, apimantis tinklo duomenis ir žurnalinius „Windows“ ir „Linux“ operacinių sistemų duomenis. Jame taip pat didžioji dalis duomenų buvo skirta tinklo lygmens atakoms (Sharafaldin et al., 2018). 2019 m. buvo pristatytas duomenų rinkinys „Mal-API-2019“ (Catak & Yazı, 2019). Jame buvo surinktos „Windows“ operacinės sistemos kenksmingo programinio kodo iškviestos sisteminių funkcijų sekos. Pagrindinis šio rinkinio minusas – jame nepateikta informacija apie funkcijų kvietimo ir grąžinamus parametrus.

Programos elgsenos modeliui nustatyti tinkamiausia yra sisteminių funkcijų sekų analizė, nes panašiai besielgiančios programos yra linkusios kviešti panašias sisteminės funkcijas. Šis metodas yra populiarus vykdant dinaminį kenksmingo programinio kodo aptikimą, kuris ir kilo iš įsibrovimo aptikimo, kai nestandartinės sisteminių funkcijų kvietimo sekos signalizuodavo apie galimus pavojus (Stephanie Forrest et al., 1998). Kadangi sisteminės funkcijos yra žemiausias bendravimo su sistema sluoksniu, jos yra pagrindinis informacijos šaltinis, leidžiantis nustatyti, kokie yra tikrieji įtartinės programos tikslai (Canzanese et al., 2016).

2. Anomalijomis pagrįsto kompiuterio lygmens įsibrovimo duomenų rinkinio kūrimas

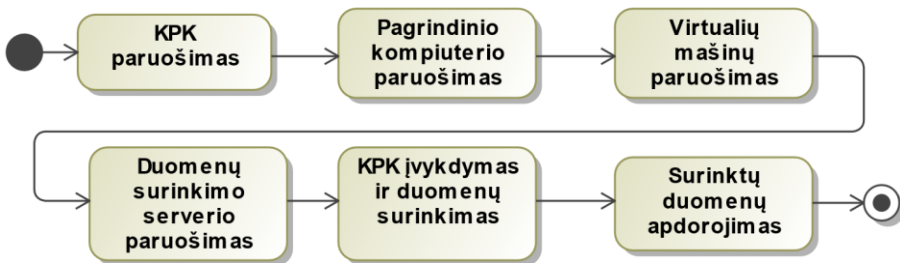
Šiame skyriuje apžvelgiamas duomenų rinkinio kūrimo metodo aprašymas ir praktinis įgyvendinimas kartu su kenksmingo kodo paruošimo aprašymu. Pristatytas duomenų rinkinys AWSCTD: dydis, struktūra, charakteristikos ir taikyti balansavimo metodai. Taip pat aptarti ir aprašyti ML/DL metodų vertinimo kriterijai, kurie bus naudojami metodų efektyvumui nustatyti vėlesniuose skyriuose.

Pristatytas duomenų kūrimo metodas buvo publikuotas tarptautiniame žurnale ir pristatytas tarptautinėje konferencijoje (Čeponis & Goranin, 2018).

Duomenų kūrimo metodas įgyvendintas naudojant laisvai prieinamus kenksmingo kodo pavyzdžius. Tai leidžia ateityje lengviau automatizuoti patį pavyzdžių parinkimo procesą. Pasiūlytas metodas turi šešis pagrindinius etapus (S2.1 pav.):

1. Kenksmingo kodo pavyzdžių paruošimas. Visų pirma, pavyzdžiai turi būti parsisiunčiami. Tada „Windows“ operacinės sistemos vykdomieji failai turi būti atrenkami ir perkelti į atskirą kolekciją.
2. Serverio paruošimas. Turi būti įdiegta ir sukonfigūruota sistemos prižiūryklė (angl. *Hypervisor*). Tada galima perkelti kenksmingo kodo pavyzdžius į paruoštą serverį.

3. Svečio mašinų paruošimas. Turi būti sukurtas ir sukonfigūruotas virtualios svečio mašinos šablonas, kuris vėliau naudojamas vykdant kenksmingą programinį kodą. Vienu metu veikiančių virtualių mašinų skaičius priklauso nuo pagrindinio serverio pajėgumų.
4. Duomenų serverio paruošimas. Turi būti paruošta vieta, skirta surinktiems žurnaliniams duomenims saugoti, tokiems kaip tinklo aktyvumas, sisteminių funkcijų sekos ir kt.
5. Kenksmingo kodo vykdymas ir duomenų surinkimas. Kai paruoštas maksimalus virtualių mašinų skaičius, galima įvykdyti pagrindinį scenarijų (angl. *script*). Jis kenksmingo kodo pavyzdį įrašo į virtualią mašiną ir ją startuoja. Tai pakartojama su visomis paruoštomis virtualiomis mašinomis. Scenarijus, įkeltas į virtualią mašiną, įvykdo kenksmingą programinį kodą ir jo veiklos žurnaliniai duomenys yra surenkami. Šie veiksmai kartojami visiems paruoštiems kenksmingo kodo pavyzdžiams.
6. Surinktų duomenų apdorojimas. Kai visi pavyzdžiai yra įvykdomi – surinkti duomenys konvertuojami į XML (arba kitą) formatą tolesnei analizei.



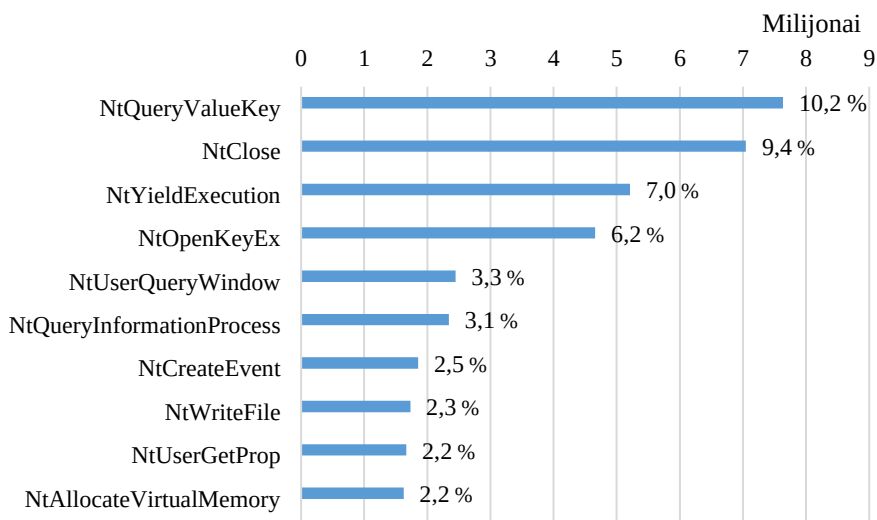
S2.1 pav. Pagrindiniai pasiūlyto duomenų rinkinio kūrimo metodo etapai

Duomenų rinkinį AWSCTD sudaro 10 276 vykdomojo tipo kenksmingo kodo pavyzdžiai iš VirusSahere.com internetinės svetainės. Papildoma informacija apie pavyzdžius buvo surinkta iš VirusTotal.com interneto svetainės.

Buvo surinkta 112,56 milijonų sisteminių funkcijų pėdsakų (angl. *traces*), kurie vėliau išsaugoti SQLite formato duomenų bazėje. Toks didelis kiekis duomenų užima 39,1 GB kietojo disko vietos. Sisteminių funkcijų sekos buvo papildytos informacija, gauta iš VirusTotal.com, naudojantis suteikta akademine prieiga. Kiekvienas kenksmingo programinio kodo pavyzdys turi sukauptą tokią papildomą informaciją:

- antivirusinių programų surinktą informaciją, pvz., tipą;
- pozityvų nuskaitymo rezultatų;
- nuorodą į kenksmingo kodo aprašą internete (VirusTotal.com puslapyje);
- kenksmingo kodo elgsenos informaciją:
 - rinkmenų išdėstymo sistemos naudojimą;
 - tinklo naudojimą (kreipiniai į lokalius ir interneto adresus);
 - naudojamos sisteminės bibliotekas (angl. *dll files*).

Labiausiai naudotos sisteminės funkcijos pateiktos S2.2 pav. Matyti, kad kenksmingas programinis kodas didžiausią dėmesį skiria „Windows“ operacinės sistemos registrų prieigai: atidaro (NtOpenKeyEx) ir skaito (NtQueryValueKey) jų reikšmes. Rinkmenų sistemos prieigos funkcijos taip pat buvo vienos iš populiariausių.



S2.2 pav. Kenksmingo programinio kodo dažniausiai naudotos sisteminės funkcijos

Nepaisant to, kad duomenų rinkinio skirtingų klasių kiekis vienodinamas (angl. *oversampling*) gana plačiai, siekiant turėti vienodą klasifikuojamų klasių skaičių, šiame darbe to nebuvo atlikta (vėlesniuose skyriuose aprašomas nekenksmingo kodo pavyzdžių pridėjimas į rinkinį) (Chawla et al., 2002). Pagrindinė priežastis – įsibrovimo ir kenksmingo kodo aptikimo uždavinyje labai svarbu turėti didesnį skaičių žalingų pavyzdžių ir svarbiausia, jie turi būti realūs, o ne sukurti dirbtinai.

Standartinės ML ir DL metodų įvertinimo metrikos yra šios: tikslumas, klasifikavimo lentelė, preciziškumas, atšaukimo statistika, F statistikos reikšmė, klaidingai neigiamų ir klaidingai teigiamų rezultatų dalys, klasifikavimo klaida, ROC kreivė, normalizuotas grupavimo įvertis (angl. *Matthews correlation coefficient*) bei apmokymo ir testavimo / klasifikavimo laikai.

3. Mašininio apmokymo metodų tinkamumo anomalijomis pagrįstam įsibrovimui aptikti ir klasifikuoti įvertinimas

Šiame skyriuje apžvelgiami eksperimentų su ML metodais rezultatai, kai apmokymui duomenimis buvo naudojamos sisteminių funkcijų sekos. Naujai pristatytam AWSCTD duomenų rinkiniui buvo pritaikyti šie ML septyni metodai: Bajeso tinklas, naivusis Bajeso

tinklas, paramos vektorių mašinos, neuroniniai tinklai, sprendimų lentelės, sprendimų medis J48 (C4.5) bei logistinis modelio medis. Skyriuje taip pat apžvelgiami vertinimo kriterijai, bandyme naudota kompiuterinė įranga ir duomenų paruošimo žingsniai. Skyriaus pabaigoje pateikiamos ML metodų eksperimento išvados. Eksperimento rezultatai buvo publikuoti tarptautiniame žurnale ir pristatyti tarptautinėje konferencijoje (Goranin & Čeponis, 2018).

Eksperimentams atlikti buvo naudojama WEKA 3.8 programinė įranga ir gana paprastos konfigūracijos kompiuteris: procesorius „Intel(R) Core (TM) i7-7500U“, 2,70 GHz (2 branduoliai, 4 gijos), 8 GB (DDR3) operatyvinė atmintis, „Windows 10 PRO“ operacinė sistema.

Buvo padaryta prielaida, kad kenksmingo kodo tipui nustatyti užtenka išanalizuoti pirmųjų sisteminių funkcijų kvietimus. Dėl šios priežasties buvo paruošti duomenys su 10, 20, 40, 60, 100, 200 ir 400 pirmosiomis sisteminėmis funkcijomis. Visi hiperparametrai naudotiems ML metodams nebuvo modifikuoti – buvo naudotos standartinės reikšmės.

Klasifikavimo tikslumo rezultatai pateikiami S3.1 lentelėje. Joje raudona spalva pažymėti blogiausi, geltona – tarpiniai, o žalia – geriausi rezultatai.

S3.1 lentelė. Klasifikavimo tikslumo rezultatai taikant ML metodus

Para- metrai	Bayes Net	Naïve Bayes	SVM	NN	Dec. Table	J48	LMT
10	87,5	61,6	89,4	87,2	88,1	88,5	89,1
20	73,8	60,9	89,4	87,8	88,0	88,8	89,2
40	74,3	61,7	91,6	88,5	89,4	90,7	91,0
60	75,0	66,1	91,8	90,2	89,3	90,9	91,2
100	75,4	61,8	92,4	91,4	90,0	92,1	92,1
200	76,1	68,2	89,8	90,3	90,0	91,9	91,8
400	75,9	69,3	87,5	84,8	89,7	92,1	92,1

Blogiausius rezultatus šiame uždavinyje parodė Bajeso šeimos metodai: Bajeso tinklas ir naivusis Bajeso tinklas. Naivusis Bajeso tinklas parodė blogiausią klasifikavimo rezultatą: 60,9 % tikslumą su 20 pirmųjų sisteminių funkcijų sekomis. SVM pasiekė patį geriausią rezultatą – 92,2 % tikslumą su 100 pirmųjų sisteminių funkcijų sekomis. Neuroniniai tinklai ir sprendimų medžio pagrindu veikiantys algoritmai geriausius rezultatus taip pat parodė su 100 pirmųjų sisteminių funkcijų sekomis. Tai parodo, kad 100 funkcijų duomenų vektorius yra optimalus, norint pasiekti didesnę nei 90 % klasifikavimo tikslumą.

Palyginus klasifikavimo laikus (S3.2 lentelė) paaiškėjo, kad geriausi rezultatai pasiekti naudojant sprendimų medžio ir sprendimų lentelės metodus. Taip pat neuroninio tinklo metodo testavimo laikas buvo 24 kartus trumpesnis nei SVM metodo.

Visi testai buvo atlikti su palyginti paprasta kompiuterine įranga: procesorius „Intel i5-4670“, 3,40 GHz. Palyginus su kitų mokslininkų naudota vaizdo plokšte TITAN X, tai yra daug silpnesnė įranga, lyginant skaičiavimo pajėgumus (Kolosnjaji et al., 2016). Taip

pat buvo taikyti paprasti ML metodai be jokio hiperparametrų optimizavimo su mažesniais duomenų vektoriais.

S3.2 lentelė. Weka pateiktos testavimo (klasifikavimo) reikšmės

Para- metrai	Bayes Net	Naïve Bayes	SVM	NN	Dec. Table	J48	LMT
10	0,003	0,009	0,130	0,002	0,002	0,005	0,003
20	0,002	0,017	0,184	0,002	0,003	0,000	0,002
40	0,003	0,031	0,309	0,009	0,000	0,002	0,003
60	0,011	0,041	0,396	0,009	0,002	0,003	0,000
100	0,019	0,078	0,586	0,024	0,000	0,000	0,003
200	0,045	0,154	1,303	0,094	0,002	0,000	0,006
400	0,100	0,454	3,098	0,413	0,005	0,000	0,016

Nepaisant pažymėtų skirtumų, pasiekti rezultatai yra geresni (Kolosnjaji et al., 2016) nei tie, kurie naudojo DL metodus. Tai leidžia teigti, kad, norint pasiekti 90 % klasifikavimo tikslumą, nėra būtina naudoti DL metodus su dideliais duomenų kiekiais. Taip pat pasiektas 90 % klasifikavimo tikslumas leidžia praktiškai taikyti tokį metodą.

4. Gilaus apmokymo metodų tinkamumo anomalijomis pagrįstam įsibrovimui aptikti ir klasifikuoti įvertinimas

Šiame skyriuje apžvelgiami eksperimentų su DL metodais rezultatai, kai apmokymo duomenimis buvo naudojamos sisteminių funkcijų sekos. Buvo naudojamas AWSCTD duomenų rinkinys. Skyriuje yra du pagrindiniai poskyriai: pirmajame aptariami paprasti DL modeliai, o antrajame – sudėtingesni DL modeliai. Šiame skyriuje taip pat aprašomas duomenų rinkinio papildymas nekenksmingų aplikacijų sisteminių funkcijų kvietimo sekomis, modelių įvertinimo kriterijų pasirinkimas, kompiuterinės įrangos konfigūracija, duomenų paruošimas bei priežastys, kodėl buvo pasirinkti tam tikri DL modeliai. Skyriaus pabaigoje pateikiama paprastų ir sudėtingų DL modelių rezultatų lyginamoji analizė ir pasiūlytas modelis, tinkamiausias kompiuterio lygmens įsibrovimui ir kenksmingam programiniam kodui aptikti. Šių eksperimentų rezultatai buvo publikuoti tarptautinėje periodinėje spaudoje (Čeponis & Goranin, 2020b; Goranin & Čeponis, 2018).

Eksperimentams su DL metodais buvo pasirinktas AWSCTD duomenų rinkinys. Artimiausias pagal duomenų tipą rinkinys yra ADFA-IDS, bet jis buvo atmestas dėl šių priežasčių:

1. Testai, atlikti su ADFA-IDS ir ML metodais, parodė prastesnius rezultatus, palyginti su gautais trečiame skyriuje (Xie et al., 2014).
2. ADFA-IDS duomenų struktūra yra panaši į AWSCTD, bet ji turi gausesnę duomenų kiekį (kenksmingo kodo pavyzdžiai ir surinktą sisteminių funkcijų pėdsakų kiekis).

3. Kadangi sisteminių funkcijų kiekis ADFA-IDS duomenų rinkinyje yra mažesnis, būtų neįmanoma atlikti eksperimentų su ilgesniais nei 600 pirmųjų sisteminių funkcijų rinkiniais.

Norint atlikti šio skyriaus eksperimentus, AWSCTD duomenų rinkinys buvo papildytas nekenksmingų aplikacijų sisteminių funkcijų pėdsakais. Jis buvo praplėstas su 16,3 milijono sisteminių funkcijų pėdsakų, surinktų iš 3 145 nekenksmingų aplikacijų (jos buvo atrinktos iš VirusShare.com svetainės ir kruopščiai sutikrintos su VirusTotal.com svetainės duomenimis, kad turėtų nulį aptikimų).

Eksperimentams su DL buvo naudojami pirmųjų programos sisteminių funkcijų kvietimo sekų rinkiniai. Kaip ir trečiame skyriuje, naudotos 10, 20, 40, 60, 80, 100, 200 ir 400 pirmųjų funkcijų. Papildomai pridėti 600, 800 ir 1000 pirmųjų funkcijų rinkiniai. Iš turimų duomenų buvo sudaryti šie rinkiniai:

- šešių klasių rinkinys (penkios kenksmingos ir viena nekenksminga klasė);
- dviejų klasių rinkinys (kenksminga ir nekenksminga klasė).

Kiekvienam iš šių rinkinių buvo padaryta po papildomą rinkinį, kuriame pasikartojantys sisteminių funkcijų numeriai sutrumpinti: jei pasikartodavo daugiau kaip tris kartus iš eilės, buvo paliekama tik po du. Pavyzdžiui: seka „4655532“ buvo paversta į „465532“. Tai buvo atlikta pagal kitų mokslininkų rekomendaciją, kuri jiems leido pasiekti geresnių klasifikavimo rezultatų (Kolosnjaji et al., 2016). Visi paruošti duomenų rinkiniai pateikti S4.1 lentelėje.

S4.1 lentelė. Apmokymo ir testavimo duomenų rinkiniai

Rinkinys	Sekų variacijos	Komentaras
AllMalware	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Tik kenksmingo kodo pavyzdžiai.
AllMalware2	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Tik kenksmingo kodo pavyzdžiai su panaikintomis pasikartojančiomis funkcijomis.
AllMalware-PlusClean	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Kenksmingo (penkios klasės) ir nekenksmingo (viena klasė) kodo pavyzdžiai.
AllMalware-PlusClean2	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Kenksmingo (penkios klasės) ir nekenksmingo (viena klasė) kodo pavyzdžiai su panaikintomis pasikartojančiomis funkcijomis.
Malware-PlusClean	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Tik dvi klasės: kenksminga ir nekenksminga.
Malware-PlusClean2	10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000	Tik dvi klasės: kenksminga ir nekenksminga su panaikintomis pasikartojančiomis funkcijomis

Ekspperimentuose naudotiems modeliams įvertinti pasitelkti šie kriterijai: tikslumas, klasifikavimo lentelė, preciziškumas, atšaukimo statistika, F statistikos reikšmė, klaidingai neigiamų bei klaidingai teigiamų rezultatų dalys, klasifikavimo klaida, ROC kreivė, normalizuotas grupavimo įvertis bei apmokymo ir testavimo / klasifikavimo laikai.

Ekspperimentui aprašyti buvo naudojama „Pytono“ programavimo kalba ir jos biblioteka „Keras“ (kartu su „Tensorflow“ biblioteka). Naudoto kompiuterio konfigūracija: procesorius „Intel i5-3570“, 3,80 GHz (4 branduoliai, 4 gijos), vaizdo plokštė GTX 1070, 16 GB operatyvinės atminties bei „Ubuntu“ operacinė sistema.

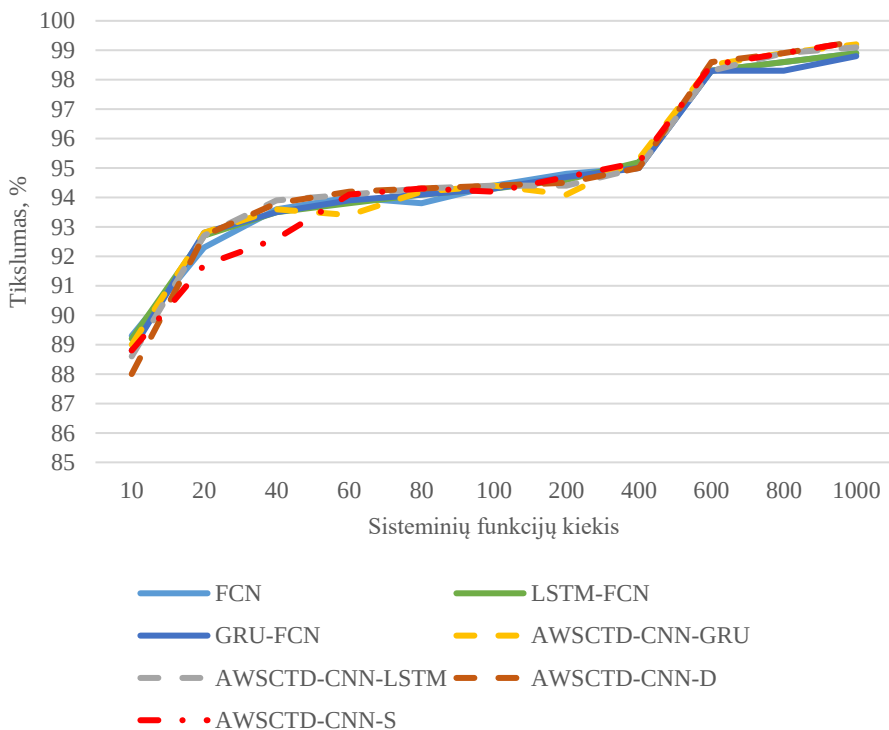
Paprastos konfigūracijos RNN ir CNN pagrindu sukurti modeliai buvo panaudoti pirmajam eksperimento daliai. Pagrindinis tyrimo tikslas – įvertinti išibrovimo aptikimą taikant DL metodus. Tam buvo naudojami du duomenų rinkiniai: „MalwarePlusClean“ (visas sisteminių funkcijų rinkinys) ir „MalwarePlusClean2“ (panaikintos pasikartojančios sisteminės funkcijos). Atlikus eksperimentą paaiškėjo, kad abiejų duomenų rinkinių šeimų tikslumo rezultatai buvo panašūs. Tai reiškia, kad, net ir sumažinus pasikartojančių sisteminių funkcijų skaičių, galima pasiekti tokius pat rezultatus kaip ir su visais rinkiniais. Geriausi tikslumo rezultatai buvo pasiekti su CNN modeliu: 94,5 %, 94,8 % ir 99,3 % tikslumas atitinkamai su 100, 400 ir 1000 pirmųjų sisteminių funkcijų. Modelio mokymo laiko eksperimentas parodė, kad ilgesnių sekų apmokymas ilgėja eksponentiškai (naudojant „AllMalwarePlusClean“ duomenis). GRU modelis demonstravo ilgiausius apmokymo laikus visiems duomenų rinkiniams. Pavyzdžiui, 400 funkcijų mokymo laikas su GRU buvo 57,7 minučių, o CNN modelis tą patį darbą atliko per 29,6 minučių. Palyginiui – 100 funkcijų mokymas su GRU truko ir CNN truko 4,6 ir 3,9 minutės atitinkamai. Tai rodo, kad 100 funkcijų sekos yra efektyviausios, kai atsižvelgiama į apmokymo laiką ir tikslumą.

Atlikus eksperimentus su paprastais DL modeliais, buvo pereita prie sudėtingų ir pažangiausių modelių (angl. *state of the art*). Tam buvo pasirinkti dviejų srautų modeliai LSTM-FCN ir GRU-FCN, nes įprastai sudėtingesnių konfigūracijų modeliai yra linkę pateikti geresnius klasifikavimo rezultatus (Szegedy et al., 2015). Abu modeliai buvo sukurti naudoti su chronologiniais duomenimis – tokio tipo yra ir AWSCTD sukauptos sisteminių funkcijų sekos. Papildomai buvo panaudoti modeliai, kurie nebuvo aprašyti anksčiau: pagrindinis tikslas juos projektuojant buvo gauti kuo mažesnę vidinių mazgų skaičių, kad apmokymo ir testavimo laikai būtų trumpiausi, bet ir kartu būtų išsaugotas aukštas klasifikavimo tikslumas. Modelių pasirinkimui įtakos turėjo kitų mokslininkų pasiekti rezultatai su kelių srautų DL modeliais (Deng & Platt, 2014; Wen et al., 2017; Yuanyuan & Zhibin, 2018). Šie papildomi DL modeliai buvo panaudoti kartu su LSTM-FCN ir GRU-FCN atliekant tolimesnius eksperimentus:

- Vieno srauto modelis, kuris sudarytas CNN sluoksnio, paimto iš LSTM-FCN, ir originaliai buvo pavadintas FCN (Zhiguang Wang et al., 2017).
- Vieno srauto CNN modelis, kuris parodė geriausius rezultatus praėjusiame eksperimente su paprastais DL modeliais. Jis pavadintas AWSCTD-CNN-D (dinaminis modelis, kurio hiperparametrų reikšmės priklauso nuo sisteminių funkcijų kiekio).
- Naujas vieno srauto CNN modelis, pavadintas AWSCTD-CNN-S (statinės hiperparametrų reikšmės).

- Naujas dviejų srautų modelis, kuris sujungia AWSCTD-CNN-D ir LSTM modelius. Jis pavadintas AWSCTD-CNN-LSTM.
- Naujas dviejų srautų modelis, kuris sujungia AWSCTD-CNN-D ir GRU modelius. Jis pavadintas AWSCTD-CNN-GRU.

Atliktų eksperimentų su sudėtingais DL modeliais tikslumo rezultatai pateikiami S4.1 paveiksle. Nė vienas iš naudotų modelių nepasiekė 90 % tikslumo su 10 sisteminių funkcijų. Šis barjeras perliptas tik su 20 funkcijų. Kitas barjeras yra 600 pirmųjų funkcijų – visi modeliai pateikė 98 % praktiškai pritaikomą klasifikavimo tikslumą. Svarbu pabrėžti, kad visi modeliai pateikė panašius rezultatus, tai reiškia, kad sudėtingi FCN šeimos modeliai nebuvo pranašesni už naujus pasiūlytus DL modelius (AWSCTD šeima).



S4.1 pav. Sudėtingų DL modelių klasifikavimo tikslumo rezultatai

AWSCTD-CNN-S modelis pademonstravo greičiausius mokymo ir testavimo laikus, kai buvo naudojami duomenys su 1000 sisteminių funkcijų. Verta paminėti, kad žymus tiek apmokymo, tiek testavimo laiko augimas prasidėdavo vienodai visiems modeliams – ties 200 sisteminių funkcijų. Atlikus geriausio modelio AWSCTD-CNN-S

įsisotinimo eksperimentą, paaiškėjo, kad klasifikavimo tikslumas, pasiektas su 1000 sisteminių funkcijų (99,3 %), yra aukščiausias ir nebekinta, jei naudojamos ilgesnės funkcijų sekos. Panaudojus papildomas 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900 ir 2000 funkcijų sekas, tikslumas buvo 99,4–99,5 %, o tai patenka į paklaidos režius.

Bendrosios išvados

1. Atlikta mokslinės literatūros analizė parodė anomalijomis pagrįstų metodų svarbumą kompiuterio lygmens įsibrovimo ir kenksmingo kodo aptikimo uždaviniuose, nes klasikiniai parašais grindžiami metodai turi tokių spragų, kaip negalėjimas aptikti nulinės dienos atakų. Šiuo metu mokslininkai yra susikoncestravę į ML ir DL metodų taikymą, bet kartu jiems trūksta kokybiškų išsamių duomenų rinkinių, skirtų „Windows“ operacinei sistemai, kurie būtų naudojami naujų metodų mokymuisi. Šis tyrimas taip pat parodė, kad sisteminių funkcijų sekos, kurios buvo iškvistos kenksmingo programinio kodo įsibrovimui atlikti kompiuterio lygmenyje, yra vienas iš perspektyviausių duomenų šaltinių. Atlikta analizė parodė, kad, norint pagerinti anomalijomis paremtų kompiuterio lygmens kenksmingo programinio kodo ir įsibrovimo aptikimą, gerinant ML ir DL metodus, reikia susikoncestruoti į naujo išsamaus kenksmingų ir nekenksmingų aplikacijų iškvistų sisteminių funkcijų sekų duomenų rinkinio sukūrimą ir tolesnį jo pritaikymą apmokant tinkamiausius ML ir DL modelius.
2. Pasiūlytas patikimas kompiuterio lygmens duomenų rinkinio kūrimo metodas paremtas kenksmingo kodo vykdymu sterilioje ir izoliuotoje virtualioje aplinkoje, kur surenkami žurnaliniai jo elgsenos duomenys. Metodas buvo įgyvendintas panaudojus nemokamą programinę įrangą ir laisvai prieinamus kenksmingo kodo pavyzdžius. Atlikti testai parodė, kad pasiūlytas metodas yra stabilus ir tinka kompiuterio lygmens duomenims surinkti. Automatinis kenksmingo kodo įvykdymas leidžia lengvai papildyti duomenų rinkinį su naujais pavyzdžiais. Kuriant duomenų rinkinį, nebuvo pastebėta jokių kenksmingo kodo vykdymo trikdžių ar jo pabėgimo iš virtualiosios aplinkos. AWSCTD duomenų rinkinys buvo surinktas panaudojus 10 276 kenksmingo kodo pavyzdžius. Šis duomenų rinkinys yra išsamesnis tiek kenksmingo programinio kodo skaičiumi, tiek surinktų sisteminių funkcijų skaičiumi. AWSCTD rinkinys taip pat turi labai svarbų pranašumą: surinktos ne tik sisteminių funkcijų sekos, bet ir joms iškviesti naudoti parametrai bei rezultatai. Naujas duomenų rinkinys taip pat buvo praplėstas panaudojant nekenksmingą programinį kodą. Naujai pristatytas duomenų rinkinys leidžia tolesnį praktinį ir mokslinį jo pritaikymą kompiuterio lygmens įsibrovimo aptikimo srityje.
3. Šiame darbe buvo pritaikyti ML metodai, norint parodyti kenksmingo programinio kodo iškvistų sisteminių funkcijų pritaikomumą IDS sugeneruotiems pranešimams prioretizuoti. Kenksmingam programiniam kodui klasifikuoti pagal šeimas buvo taikyti septyni ML metodai. Rezultatai parodė, kad šis būdas gali būti naudojamas praktiškai – geriausias 92,4 % klasifikavimo tikslumas buvo pademonstruotas taikant SVM metodą su 100 pirmųjų sisteminių funkcijų. Panašūs rezultatai buvo pasiekti panaudojant J48 ir LMT metodus. Jie parodė

- 92,1 % klasifikavimo tikslumą. J48 metodas taip pat pademonstravo trumpiausius mokymo ir klasifikavimo laikus, kurie yra labai svarbūs, kai nauji pavyzdžiai turi būti įvedami į modelį. Nepaisant pasiekto gana aukšto 90–92 % klasifikavimo tikslumo (tinkamo praktiškai naudoti kaip papildoma analizė yra atliekama ekspertų) bei trumpo apmokymo laiko, taikant ML metodus, tolesni tyrimai su DL reikalingi norint sumažinti klaidingai neigiamų rezultatų skaičių.
4. Lyginamoji paprastų DL modelių analizė atlikta norint įvertinti jų efektyvumą, klasifikuojant kenksmingą programinį kodą bei atskiriant kenksmingas veikas nuo palankių. Palyginus su kitų mokslininkų pasiektais klasifikavimo ir įsibrovimo aptikimo rezultatais, AWSCTD duomenų rinkinys juos pagerino 13–38 %. Pasiektas didesnis nei 90 % tikslumas leidžia taikyti DL metodus hibridiniuose arba didelėms įmonėms skirtuose saugumo įrankiuose, kurie sujungia automatinį didžiosios dalies anomalijų aptikimą ir ekspertų atliekamą neišščių atvejų analizę. Visi trys modeliai pagerino paprastų NN ir SVM modelių rezultatus, kai buvo naudojamos ilgesnės sisteminių funkcijų sekos. Geriausi rezultatai buvo pasiekti CNN modelio: didesnis nei 93,0 % tikslumas atliekant šeimos klasifikavimą ir 99,3 % tikslumas atliekant įsibrovimo aptikimą. CNN modelis pralenkia sekomis pagrįstus LSTM ir GRU modelius visomis metrikomis. Pasikartojančių sisteminių funkcijų mažinimo metodas turėjo teigiamą įtaką eksperimento rezultatams. Ilgesnių sisteminių funkcijų sekų naudojimas turėjo neigiamą įtaką metodų apmokymui – trukmė ilgėjo eksponentiškai. Buvo pastebėta atvirkštinė epochų įsisotinimo priklausomybė nuo sisteminių funkcijų sekų ilgio.
 5. Eksperimentai, atlikti su septyniais sudėtingais DL modeliais, nepademonstravo jokio dviejų srautų modelių pranašumo, lyginant su vieno srauto modeliais, kai buvo atliekama kompiuterio lygmens įsibrovimo ir kenksmingo programinio kodo atpažinimo užduotis. Dviejų srautų modelių pasiekta 99,2 % tikslumas buvo panašus į paprastų vieno srauto modelių pasiektą rezultatą (99,3 %), turint omenyje, kad 1000 funkcijų sekų klasifikavimas pateikė 1 % paklaidą. Dviejų srautų modelių efektyvumo įverčių rezultatai buvo panašūs su vieno srauto modeliais, bet klasifikavimo ir testavimo laikų rezultatai buvo prastesni. Tai yra svarbu atliekant įsibrovimo ir kenksmingo programinio kodo aptikimo užduotį: dviejų srautų modeliai buvo nuo 3 iki 7 kartų lėtesni, naudojo daugiau kompiuterio resursų ir reikalavo daugiau epochų apmokymo įsisotinimui pasiekti, palyginus su geriausiai pasirodžiusiu statiniu vieno srauto modeliu. Visi išmėginti DL modeliai sugebėjo pasiekti 90 % tikslumą su 20 funkcijų sekomis ir praktiškai pritaikomą 98 % tikslumą su 600 funkcijų sekomis. Geriausia klaidingai teigiamų rezultatų reikšmė (maždaug 2 %) buvo pasiekta panaudojus AWSCTD-CNN-GRU modelį, o geriausia klaidingų neigiamų rezultatų reikšmė buvo pasiekta AWSCTD-CNN-S ir AWSCTD-CNN-D modelių ir buvo mažesnė nei 1 %. Naujai pasiūlytas AWSCTD-CNN-S modelis gali būti svarstomas kaip nugalėtojas, kuris pademonstravo geriausią tikslumą (99,3 %), trumpiausius mokymo ir testavimo laikus ir geriausią klaidingų neigiamų rezultatų reikšmę. Pasiektas didelis klasifikavimo tikslumas leidžia naudoti šį modelį saugos sprendimuose su pagalbiniais ekspertais ir 600 pirmųjų sisteminių funkcijų bei visiškai automatizuotą pritaikymą sistemose naudojant 1000 sisteminių funkcijų.

Annexes¹

- Annex A.** The Result Tables of Deep Learning Experiments
- Annex B.** Declaration of Academic Integrity
- Annex C.** The Co-Authors' Agreements to Present Publications Material in the Dissertation
- Annex D.** Copies of Scientific Publications by the Author on the Topic of the Dissertation

¹ The annexes are supplied in the enclosed compact disc.

Dainius ČEPONIS

RESEARCH OF MACHINE AND DEEP LEARNING METHODS
APPLICATION FOR HOST-LEVEL INTRUSION DETECTION
AND CLASSIFICATION

Doctoral Dissertation

Technological Sciences,
Informatics Engineering (T 007)

MAŠININIO IR GILIAUS APMOKYMO METODŲ
TAIKYMO ĮSILAUŽIMŲ KOMPIUTERIO LYGMENYJE APTIKIMUI
IR KLASIFIKAVIMUI TYRIMAS

Daktaro disertacija

Technologijos mokslai,
informatikos inžinerija (T 007)

2021 04 12. 15,75 sp. l. Tiražas 20 egz.
Leidinio el. versija <https://doi.org/10.20334/2021-008-M>
Vilniaus Gedimino technikos universitetas
Saulėtekio al. 11, 10223 Vilnius
Spausdino BĮ UAB „Baltijos kopija“
Kareivių g. 13B, 09109 Vilnius

Annexes

Annex A. The Result Tables of Deep Learning Experiments

Table A.1. Malware type classification with the help of DL and SVM methods (AllMalware set)

	Accuracy (percent)				Classification time (milliseconds)			
Count	LSTM	GRU	CNN	SVM	LSTM	GRU	CNN	SVM
10	88.1	88.3	87.3	89.4	0.0926	0.084	0.0401	0.044
20	88.8	88.1	89	89.4	0.1043	0.0994	0.051	0.0583
40	89.1	90.6	91.2	91.6	0.1327	0.1377	0.0618	0.0842
60	88.2	90.5	91.2	91.9	0.1786	0.1704	0.089	0.0848
80	91.8	91.6	92.3	92.7	0.2194	0.2221	0.1157	0.0937
100	91.6	91.9	92.8	92.4	0.2559	0.2566	0.129	0.1165
200	90.4	91.5	92.7	89.6	0.444	0.4363	0.3019	0.2392
400	87.6	90.3	93	87.3	0.984	0.8142	0.6222	0.6739
600	87.4	91.4	93.1	86.1	2.3443	2.3052	1.6681	1.5564

End of the Table A.1

	Accuracy (percent)				Classification time (milliseconds)			
Count	LSTM	GRU	CNN	SVM	LSTM	GRU	CNN	SVM
800	82.1	88.5	93	84.7	4.3159	3.7894	2.3929	2.2612
1000	75.5	89.6	93.1	83.2	6.8075	5.6159	3.3276	3.2245

Table A.2. All malware plus clean samples accuracy results. A total of six classes was classified

	AllMalwarePlusClean				AllMalwarePlusClean2			
Count	LSTM	GRU	CNN	SVM	LSTM	GRU	CNN	SVM
10	77.2	78.3	78.7	79.9	80.1	79.4	80.1	81.2
20	83	82.3	83.4	83.6	85.8	85.6	86.9	86.7
40	84.1	84.1	85.1	85.3	87.3	87.6	88.8	88
60	85	85.8	86.1	86.2	87.5	86.4	88.5	88.2
80	87.1	86.8	88.5	87.7	87.5	88	89.2	87.9
100	87.1	86.5	88.2	87.4	87.7	87.9	88.8	87.6
200	85.8	87.5	89.4	86.5	86.2	87.3	89.6	86.3
400	80.9	88.1	89.9	81.9	80.3	87.3	89.6	78.1
600	79.8	89.1	93.2	75.5	77.6	89.4	93.5	74.8
800	68.3	85.3	93.5	73.9	41.5	92.3	93.6	73.3
1000	77.1	89.1	93.6	72.9	64	84.1	93.9	72.3

Table A.3. Malicious and benign samples (Malware or Clean) classification accuracy

	MalwarePlusClean				MalwarePlusClean2			
Count	LSTM	GRU	CNN	SVM	LSTM	GRU	CNN	SVM
10	87	87.4	87.4	87.4	88.6	88.6	88.9	88.4
20	90	90.2	91.1	90.2	92	92.4	93	91.8
40	91.2	90.6	91.8	90.5	92.9	92.4	94.1	92
60	91.2	91.1	93.3	91.1	93.2	93	93.5	91.9
80	92.7	92	94.2	91.5	93	93.2	94.5	91.7
100	92.7	92.5	94.3	91.5	93	92.1	94.5	91.2

End of the Table A.3

	MalwarePlusClean				MalwarePlusClean2			
Count	LSTM	GRU	CNN	SVM	LSTM	GRU	CNN	SVM
200	90.7	90.8	94.3	88.6	92.4	91.8	94.9	88.9
400	87	90.9	94.8	88.4	89.2	92	94.8	88.7
600	86.4	88.7	98.5	87.2	85.8	90.1	98.6	87.1
800	84.2	84.7	98.8	86.9	85.1	82.8	98.9	87
1000	84.2	70.3	98.9	87.2	83.8	72.4	99.3	87

Table A.4. Classification accuracy in percent

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	88.8	88.0	88.6	89.0	89.3	89.2	88.8
20	91.7	92.7	92.7	92.8	92.3	92.7	92.8
40	92.6	93.8	93.9	93.6	93.6	93.5	93.5
60	94.1	94.2	94.1	93.4	94.0	93.8	93.9
80	94.3	94.3	94.3	94.2	93.8	94.1	94.1
100	94.2	94.4	94.4	94.4	94.4	94.3	94.3
200	94.7	94.5	94.4	94.1	94.8	94.6	94.7
400	95.2	95.0	95.0	95.3	95.0	95.2	95.0
600	98.5	98.6	98.3	98.5	98.3	98.3	98.3
800	98.9	98.9	98.9	98.9	98.6	98.6	98.3
1000	99.3	99.3	99.1	99.2	98.9	98.9	98.8

Table A.5. Training time for one-fold (seconds)

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	30.11	52.09	91.47	81.92	79.56	106.32	102.61
20	44.80	60.18	94.96	97.15	101.43	118.97	122.03
40	43.73	61.18	105.64	115.05	98.51	161.69	137.87

End of the Table A.5

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
60	51.94	57.38	125.74	129.02	126.90	166.86	148.37
80	63.99	66.29	149.29	144.12	125.75	189.16	162.97
100	82.33	72.16	140.06	128.47	128.59	182.53	187.53
200	100.24	98.69	161.95	197.37	142.61	285.58	248.22
400	147.46	217.77	684.11	256.65	293.85	588.43	465.05
600	279.19	363.71	1190.30	578.03	280.22	1191.68	601.33
800	507.41	852.84	2246.27	887.89	565.09	2059.99	794.29
1000	453.21	745.45	2785.16	1043.92	549.20	3256.41	959.96

Table A.6. Classification time for one sample (seconds)

Count	AWS-CTD-CNN-S	AWS-CTD-CNN-D	AWS-CTD-CNN-LSTM	AWS-CTD-CNN-GRU	FCN	LSTM-FCN	GRU-FCN
10	0.00004	0.00004	0.00007	0.00007	0.00011	0.00014	0.00013
20	0.00005	0.00005	0.00009	0.00009	0.00011	0.00015	0.00014
40	0.00007	0.00007	0.00012	0.00013	0.00014	0.00018	0.00018
60	0.00010	0.00009	0.00017	0.00017	0.00016	0.00024	0.00021
80	0.00012	0.00012	0.00021	0.00021	0.00019	0.00027	0.00025
100	0.00016	0.00015	0.00023	0.00025	0.00021	0.00031	0.00029
200	0.00026	0.00030	0.00042	0.00039	0.00034	0.00053	0.00047
400	0.00055	0.00063	0.00123	0.00084	0.00066	0.00128	0.00085
600	0.00138	0.00164	0.00296	0.00172	0.00139	0.00283	0.00173
800	0.00182	0.00218	0.00479	0.00247	0.00194	0.00461	0.00229
1000	0.00223	0.00312	0.00770	0.00347	0.00248	0.00710	0.00283

Annex B. Declaration of Academic Integrity

Annex C. The Co-Authors' Agreements to Present Publications Material in the Dissertation

Annex D. Copies of Scientific Publications by the Author on the Topic of the Dissertation