International Journal of Information Security (2023) 22:1893–1919
https://doi.org/10.1007/s10207-023-00725-8

REGULAR CONTRIBUTION

On the detection of lateral movement through supervised machine learning and an open-source tool to create turnkey datasets from Sysmon logs

Christos Smiliotopoulos[1] · Georgios Kambourakis[1] · Konstantia Barbatsalou[1]

Published online: 19 July 2023

Abstract
Lateral movement (LM) is a principal, increasingly common, tactic in the arsenal of advanced persistent threat (APT) groups and other less or more powerful threat actors. It concerns techniques that enable a cyberattacker, after establishing a foothold, to maintain ongoing access and penetrate further into a network in quest of prized booty. This is done by moving through the infiltrated network and gaining elevated privileges using an assortment of tools. Concentrating on the MS Windows platform, this work provides the first to our knowledge holistic methodology supported by an abundance of experimental results towards the detection of LM via supervised machine learning (ML) techniques. We specifically detail feature selection, data preprocessing, and feature importance processes, and elaborate on the configuration of the ML models used. A plethora of ML techniques are assessed, including 10 base estimators, one ensemble meta-estimator, and five deep learning models. Vis-à-vis the relevant literature, and by considering a highly unbalanced dataset and a multiclass classification problem, we report superior scores in terms of the $F1$ and AUC metrics, 99.41% and 99.84%, respectively. Last but not least, as a side contribution, we offer a publicly available, open-source tool, which can convert Windows system monitor logs to turnkey datasets, ready to be fed into ML models.

Keywords Lateral movement · Sysmon · Dataset · Attacks · Network security · Machine learning

1 Introduction

In recent years, numerous individuals, organizations, and government bodies have suffered from repeated incidents of lateral movement (LM). Sensitive data have been stolen or lost, including bank accounts, fighter aircraft blueprints, or even classified state secrets as part of an international information leakage cyberattack. Generally, LM refers to the broader field of the application of malicious techniques that adversaries exploit to acquire unauthorized access through a network's endpoint towards the lateral escalation of their privileges in search of critical infrastructures to compromise and the exfiltration of valuable data [1]. Simply put, the attacker's goal is to gain an initial foothold in a networking environment, remain undetected for as long as it is demanded for learning the targeted facilities' topology, maintain ongoing access by moving laterally through the compromised environment, and finally elevate its privileges towards data extraction or elimination. LM tactics are categorized within the general area of advanced persistent threats (APTs) [2]; colloquially, it is the act of acquiring as much network access as possible, mostly to achieve persistence.

LM should be distinguished from the legacy cyberattacks of the past and considered more as key tactics, unbounded to specialized tools. A typical LM technique comprises three major stages, namely the reconnaissance and enumeration of the targeted computing facility, the credential dumping and privilege escalation, and finally the compromising of the targeted device. Precisely, during reconnaissance and enumeration, the adversary explores through mapping the network's topology, devices, operating systems and user's hierarchy. Privilege escalation is then accomplished

✉ Christos Smiliotopoulos
csmiliotopoulos@aegean.gr

Georgios Kambourakis
gkamb@aegean.gr

Konstantia Barbatsalou
tbarbatsalou@gmail.com

[1] Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Karlovasi, Samos, Greece

with credential dumping though a large variety of hashing exploitation techniques and towards the final goal of compromising valuable assets. Pivoting is tightly related to the concept of LM, and in some contexts, these terms are used interchangeably. Nevertheless, pivoting is more precisely used to refer to the act of moving from host to host inside the target network, while LM also entails the act of privilege escalation on the compromised machines.

Conducting LM is a trademark of contemporary and sophisticated threat actors, as that is evidenced by MITRE's ATT&CK Framework records of common LM techniques [3]. LM tactics are recognized as "TA0008" in MITRE's list, which is constantly updated with the most impactful incidents and APT groups that conducted them. Prominent threat actors in this context include the APT39 cyber espionage group [4] that is alleged to be responsible for numerous thefts of personal information around the world and the APT29 group (aka "Cozy Bear"), which is reported to be behind the infamous compromise of the SolarWind's "Orion" network monitoring software [5]. The impact of LM events around the world is so repeatedly ominous that VMware's 2022 "Global Incident Response Threat Report" [6], revealed that LM tactics were used in 25% of all the reported attacks. Although a calibrated to LM detection endpoint detection and response (EDR) policy may be effective to some extent, mainly due to the immense volume of network traffic and audit logs, the solution lies in the introduction of a log-based intrusion detection system (IDS) that leverages contemporary machine learning (ML) techniques.

In this context, after pinpointing the shortcomings of the relevant literature, the current work delivers a multifold novel contribution regarding the ecosystem of LM IDS by means of supervised ML techniques. Concentrating on the MS Windows platform and its system service known as system monitor (Sysmon), we provide a comprehensive supervised classification methodology that involves an abundance of traditional (shallow learning) classifiers, both base estimators and ensemble meta-estimators, and deep learning (DL) models. This allows for a comprehensive picture of this potential and sets the basis for future research in this timely domain. Especially, regarding the research methodology, we explicate and contextualize the feature selection, data preprocessing, and feature importance processes, and delve into the ML model parameterization, including hyperparameters. On top of that, we offer an all-encompassing solution to generate labeled or unlabeled CSV datasets from voluminous Sysmon logs. Overall, the key contributions of this work vis-á-vis the relevant literature can be outlined as follows:

- We detail the hurdles involved in the creation of turnkey unlabeled or labeled datasets in CSV format through the manipulation of EVTX Sysmon logs, and propose a software solution able to automatize this task. This contribution is key to the LM ecosystem given that, to the best of our knowledge, no pertinent datasets exist, obstructing research on ML-oriented LM detection.
- We provide a detailed overarching methodology behind human-driven feature selection upon Sysmon log-based datasets. The classification features outlined can be used as a solid reasoning to the creation of robust and potentially high-rated IDS targeting LM. The suggested methodology is full-fledged, ranging from the labelling and preprocessing of data to the selection of the most applicable per ML model hyperparameters.
- Differently from the existing literature, we formulate a multiclass problem, meticulously assessing the proposed methodology through a great variety of legacy classifiers and DL models.
- We provide a scrupulous review of the relevant literature, also vis-à-vis our work, pinpointing misconceptions and dubious practices.

The remainder of this paper is structured as follows. The next section provides an overview of the related work. Section 3 focuses on the obstacles related to harnessing Sysmon logs for ML-powered intrusion detection, and details a solution to this end. The same section outlines the dataset used in the context of this work. Our methodology, including feature selection, data preprocessing, and feature importance, is given in Sect. 4. The setup and results of the experiments are presented in Sect. 5, followed by an in-depth discussion in Sect. 6. The last section concludes and provides pointers to future work. For easier guidance throughout the manuscript, a list of abbreviations is included at the end of the article.

2 Related work

The current section provides a brief review of the key pertinent literature regarding LM. The concentration is on the methodology of each relevant work concerning the detection of LM through either supervised or unsupervised machine learning techniques or graph-based analysis. That is, although the work at hand deals with the identification of LM by means of supervised learning, for reasons of completeness, the current section presents the related literature for all the three aforementioned categories. It is to be noted that a more detailed, focused on particular aspects, comparison with the related work is given in Sect. 6.3. For easy reference, the key characteristics of every work discussed in this section are summarized in Table 1.

2.1 Supervised learning based schemes

Based on its impact, the work in [7] is considered a state of the art regarding the subject of anomaly detection through security logs. Specifically, the authors propose an anomaly detection approach that is based on a mixture of 10 log-based generic features and eight custom-made others, respectively. Both set of features were extracted from the publicly available Los Alamos National Laboratory (LANL) dataset collected between 1996 and 2005 [8]. Sampling techniques were applied on the collected subset to facilitate processing and computational power issues with such large data volumes. Supervised ML techniques, namely, Random Forest (RF), LogitBoost (LB) and Logistic Regression (LoR) were implemented towards the classification of the identified log events into normal or malicious. The performance of the classifiers was evaluated against the false positive (FPR) and false negative (FNR) rates, while the malicious authentication predictions of the three aforementioned classifiers were fed to the ensemble Majority Voting uniform weighted algorithm and re-evaluated. The authors give no insight regarding their understanding and the extended graphical-based experimentation upon the dataset that led to the extraction of the presented composite features. Additionally, they do not provide any of the implemented R language scripts, obstructing reproducibility.

The authors in [9] introduced a hybrid anomaly detection approach, focusing on the identification of networking hosts susceptible to LM techniques during the early stages of their exposure to the threat. The first part of their work is dedicated to the graphical representation of authentication logs included in the LANL dataset towards the extraction of 29 composite features. Above that, six more flow-based features were extracted from the relevant to the network flow event-logs. In the second part, the 35 finally extracted features were evaluated under several supervised classifiers, namely, Decision Tree (DT), RF, Linear Regression (LiR), Gaussian Naive Bayes (GNB) and Label Binarizer (LaBi), as part of the proposed anomaly-based approach. Under and oversampling techniques were applied on the dataset due to its highly imbalanced nature, as long as k-fold cross validation (k=10) during the execution of each ML technique. Interestingly, the same work [9] was revisited in [10] under a case study concentrating on RDP-based LM techniques. Keeping the same principles as in [9], the authors leveraged Windows host-based RDP event logs (as evidences) through the combination of two publicly available Windows event-logs subsets of the LANL dataset, namely, "comprehensive" and "unified", respectively. The subject of LM detection via authentication logs was re-addressed in [11] by the same authors, although extended in the examination of the effects on classification efficiency due to perturbations of LM techniques patterns.

Moreover, the work in [12] introduced a Sysmon log-based anomaly detection system based on shallow and deep neural networks (DNN) supervised techniques, namely, LSTM, RNN, and SVM. On top of that, the authors proposed a generic set of features based on the manipulation of Sysmon EventIDs and evaluated their scheme in terms of TP and TN rates.

Despite the promisingly presented results in [9–12], the hybrid-combined dataset was not made publicly available. Moreover, the superiority of the classification results against the one in [7] was fully documented only in [11] through the representation of the ROC-curve and the precision, recall, and $F1$-score ($F1$) metrics. On the other hand, in [9, 10, 12] the authors neglected to mention the criteria upon which their claims over the work in [7] were based.

2.2 Unsupervised learning based schemes

So far, only a few works considered unsupervised ML as a means for the evaluation of a sparse diversity of collected logs exclusively related to LM. The examined features were either generically fundamental to the initially analyzed log-based datasets or manually extracted from the various interrelated nodes and edges representing the topology of the network. Precisely, the work in [13] proposed an anomaly detection method that was based on ensemble unsupervised ML to identify traces on compromised hosts with LM techniques. The authors used the LANL dataset [8] to create a graph-based model, which depicts the various communications between the targeted hosts. The classification features were extracted and evaluated with an ensemble of unsupervised ML techniques, namely principal component analysis (PCA), k-means clustering, and median absolute deviation-based outlier (MADO) detection. The method's accuracy was evaluated under a trace-related simulation case study.

In the same context, the authors in [14] employed four unsupervised ML methods, namely Autoencoder (AE), Isolation Forest (IF), lightweight on-line detection of anomalies (LODA) and local outlier factor (LOF), under an anomaly detection scheme which targets the identification of insider attacks. Various preprocessing techniques were applied on data with temporal payload to fit with deep learning (DL) algorithms and contribute in revealing patterns of adversarial changes in user's behavior. Unsupervised ML ensembles were created to evaluate the anomaly detection performance under different algorithmic combinations. The results were compared against several state-of-the-art works using well-known datasets, including CERT [15], LANL [8] and TWOS [16]. On the downside, both the works in [13, 14] lack of experimental feedback from real-world data stemming from LM enterprise scenarios and events.

An almost similar to the works in [13, 14] hybrid approach was presented in [17]. The theories of network embedding, for mapping a network's graphical representation into nodes and vectors, were mixed with feature aggregation techniques towards the formation of composite features. The authors evaluated the finally selected features under a proposed semi-supervised classification algorithm based on the Denoising autoencoder unsupervised model. The experiments were conducted on a balanced subset of the LANL dataset [8] that is called "The Comprehensive, Multi-Source, Cyber-Security Event" and the final classification results were evaluated under FPR, TPR, accuracy (ACC) and precision (PREC) metrics. Although the authors presented an estimated ACC of 99.9% with 91.3% PREC on a ratio of 10% of labelled data, their outcomes represent only the ideal situation of a balanced dataset in a lab-oriented pretentious way, and it is hardly applicable to real-life unbalanced data.

Additionally, the work in [18] considered the detection of malware LM on data centers though the implementation of a behavioral unsupervised ML model. Anomaly detection was conducted on the application layer network traffic of data centers via the Jaccard Similarity Coefficient and clustering measurement technique (JSCC) on several balanced datasets. On the other hand, the authors in [19] presented an unsupervised learning model of LM detection, based on the role-based approach of clustering the system connections to remote hosts into distinct roles. We argue that the type of traffic, and therefore the features obtained in both the environments of [18, 19], are significantly different compared to this study. Namely, the traffic considered by the authors is inappropriate for detecting LM. Therefore, such works are considered out of scope of the current study.

2.3 Graph-based schemes

The authors in [20] address the subject of LM detection through the definition of a graph-based impact metric. First, the evolution of the various paths, that an adversary could take among the various network nodes due to the exploitation of various vulnerabilities, is defined algorithmically via the introduction of a dynamic graph-based reachability model (DGBR). This model is then used as the basis for the calculation of a network-level impact score. The latter score is quantified based on the value and reachability score assigned to each network node that could be compromised by adversaries. Although the proposed model was implemented in the context of the so-called Windows credentials "Pass-the-Hash" (PtH) vulnerability, the authors do not consistently abide by their defined model. Instead, new concepts were introduced which lack sufficient documentation and connection with the already presented theory. Besides that, the case study scenario of PtH that was tested against the LANL dataset was based on the implementation of the proposed reachability and impact metric model on C++ source code that was not included, making replication of the experiments practically infeasible.

The work in [21] contributed a graph-based detection system, dubbed Latte, which deals in parallel with the multi-layered nature of large-scaled data stemming from LM incidents and the lack of knowledge regarding adversaries, respectively. They address the LM problem in two ways. First, hosts and user accounts were marked as nodes, while their interconnections were modeled as edges. Once an infected node is identified, it leads through the proposed forensic algorithmic analysis to any other compromised element(s). Second, a general algorithmic approach of rare paths anomalies identification leverages a remote file execution detector to recognize unknown LM attempts. The same work [21] inspired two more similar approaches. Precisely, the authors in [22] presented another tool, dubbed Hopper, that is concentrated on the identification of malicious LM events through real-life collected logs. The proposed system, tracks user's login activities and outlines their correlations among hosts on a graph-based representation. The process ends with the detection of anomalies in login patterns, which may imply the existence of LM. Moreover, the authors in [23] introduced a custom LM detection algorithm under the title LMTracker. This scheme originated from the authors' effort to address the gaps in the efficiency of the existing endpoint protection practices to identify LM events. Various elements included in the captured log-based traffic, namely users, computers, processes etc., were extracted and implemented as nodes for the construction of heterogeneous graphs that present the various relationships among its elements. In turn, the advanced graph neural networks theory was used for the production of two custom algorithms for the representation of the LM-related paths and the unsupervised anomaly path detection based on a predefined threshold, respectively. LMTracker was evaluated over LANL [8] and CERT 6.2 [15] datasets, and the experimental results were examined under the prism of confusion matrix rates and the ROC-AUC metric. While highly promising as an LM anomaly detection tool with approximately 0.95 ROC AUC score, the LMTracker presents noticeable FP rates.

2.4 Key observations

With reference to Sects. 2.1 to 2.3 and Table 1, almost half of the presented works (namely 5 out of 12) relied on supervised shallow classifiers, whereas from the rest two categories three contributions implemented unsupervised classification techniques and other four were based on graphs. Above that, a characteristic common to most works is that they neither construct their own set of data logs and samples, nor provide adequate reference to regularization techniques and hyperparameter optimization steps. Interestingly, all the works but two have been published from 2018 onward.

Furthermore, the vast majority of the works in Table 1 utilized logs collected as public via the Windows Event Viewer tool, all of which were related to the legacy LANL dataset of multi-source cyber-security events. Released as public in 2015, LANL is considered almost outdated due to its noninclusion in samples derived from contemporary malicious techniques, besides LM traffic. Precisely the small proportion of the included malicious traffic, led most of the authors to reproduce artificially the aforesaid samples in order to create as custom an adequate to be manipulated with ML techniques imbalanced dataset. We argue that similar, to the aforesaid, processes related to artificial data handling and manipulation of datasets should be cared with great concern. In most of the cases, they are not related to real-life traffic and may mislead the prediction rates of the whole ML-IDS process, despite the good results that may initially reveal. Furthermore, even the works in [9–12], that introduced a different to LANL hybrid-combined dataset neglected to provide it, obstructing reproducibility. Another important aspect that needs to be pointed out is that most of the works presented in Sects. 2.1 to 2.3 neither mention the selected for the classification process features nor justify their contribution to the whole ML process. Further, no code regarding the Python or R implemented scripts is provided, not to mention the lack of hyperparameters upon which the ML models were constructed.

All in all, a general conclusion is that the majority of the studies so far have been conducted on datasets that do not meet a number of criteria, namely contemporary LM or general purpose attacks, adequate representation of all the included classes to help the ML experimental process or even multiclass labelling of the included samples. Another important observation is that all but one [12] of the works relied on MS Windows event viewer collected logs and none of them introduced Sysmon related traffic to take advantage of the enhanced headers as those are precluded by the collected event-logs. We argue that this phenomenon is mainly due to the lack of an open-source, publicly available tool able to readily convert Sysmon's extracted logs (EVTX format), to a (un)labeled dataset in CSV format. This shortcoming is also obvious in other recent works which rely on manual investigation of log files produced by Windows Event Viewer, and for that reason the preprocessed in comma-separated format LANL was selected in most of the cases.

As it concerns Sysmon logs manipulation, the work in [24], was the first to deal with the presentation of a LM-oriented EDR policy towards the first level identification of LM incidents thought the analysis of raw log files. The work ended with the presentation and evaluation of the Python Evtx Analyzer (PeX) EDR tool, which incorporated the aforementioned EDR-policy's criteria. The tool manipulates Sysmon files in their raw EVTX form, which are then iterated over the presented EDR policy's features to reveal the existence of potential malicious LM activity. The PeX tool is publicly available on GitHub [25].

As an extension to [24], and for addressing the key gap of the creation of datasets through EVTX log files, among others, the current work contributes such a tool, entitled evtx_To_CSV_Export Tool (ETCExp). The tool, detailed in Sect. 3, was developed to serve as an easily configurable and above all OS-independent command line tool that helps incident response teams and researchers to parse and transform massive EVTX log files into compatible unlabeled datasets (CSV files), ready to be used along with ML algorithms. Further, ETCExp tool is designed to implement the proposed in [24] EDR policy for automatically labelling the transformed Sysmon logs, into a multiclass CSV set of samples. Besides the labelling process, the ETCExp tool performs on demand, feature selection, subsets extraction, and basic data preprocessing through One Hot Encoding and MinMax algorithms. The full presentation of the technical characteristics of the aforesaid tool is available in Sect. 3.

3 ETCExp: converting Sysmon logs to CSV

MS Windows Sysmon logs consist a powerful source for LM technique's indicators that can be used as inputs within elevated ML-IDS analysis techniques. Nevertheless, the manipulation of big volume log files as part of an EDR workflow in the context of incident response teams is a demanding procedure. Originally, EDR teams handled the limited number of log-traffic's headers derived from MS Windows Event Viewer logger, but nowadays Sysmon is the prominent, modern substitute. On the downside, the great variety of generic and artificially created log headers of Sysmon multiplied the diversity and difficulty level regarding the analysis of log-based traffic. As discussed in the previous section, a key gap in the literature of LM detection is that until now there is no open-source, lightweight, publicly available tool to manipulate Sysmons' EVTX files and convert them to an equivalent CSV format, ready to be fed to ML classifiers. To serve this key need, this section details on the creation of such a tool, and through it, it provides a proof of concept (PoC) dataset, which is used in the subsequent sections to form an ML-focused detection methodology and evaluate LM intrusion predictions via both shallow and deep ML models.

3.1 Preliminaries

Sysmon is a multipurpose service of the MS Windows OS environment and a system's driver too [24, 26]. It is not pre-installed on any Windows OS version, but after imported it remains omnipresent with all Windows internal tasks, including reboots and event logging information (namely process creations, network connections and modification actions among many others). Sysmon monitors and gathers detailed event-oriented information that is organized in 27 distinct types of case-sensitive EventIDs, as presented in [26], including logs from system processes, networking com-

Table 1 Summary of the key aspects of the works included in this section. The works are arranged in chronological ascending order

Related work			
Title	Year	Method	Summary
A graph-based impact metric for mitigating lateral movement cyber attacks [20]	2016	DGBR model	Introduction of a DGBR model that keeps track of the various adversarial paths that may be followed during the exploitation of vulnerabilities with LM techniques. Through the DGBR model, the authors calculate a network-level impact score that is implemented along with several other algorithmic functions, towards the evaluation of a PtH case study on the LANL dataset
An unsupervised multi-detector approach for identifying malicious lateral movement [13]	2017	UML (PCA, k-means, MADO, Ensemble ML)	Automated unsupervised anomaly detection ensemble method towards the identification of LM traces on infected hosts. The LANL dataset is used for the log-based graphical representation and the extraction of features. The extracted data are evaluated via two independent anomaly detection methods, which incorporate PCA, k-means and MADO techniques, respectively. The results of the two methods are combined and re-evaluated under a parameter-based ensemble learning method
A novel approach for identifying lateral movement attacks based on network embedding [17]	2018	Network Embedding, Denoising Autoencoders	Semi-supervised classification through Denoising autoencoder of features emerged from network embedding and feature aggregation techniques
Detecting malicious authentication events trustfully [7]	2018	SSC (RF, LB, LoR, MV)	A log-based anomaly detection approach applied on generic and custom-made/engineered (artificial/synthetic) log features that are extracted from the LANL dataset. The application of sampling techniques precedes the samples' classification with SSC ML techniques in normal or malicious, while the final results are evaluated under the FPR and FNR metrics
Latte: large-scale lateral movement detection [21]	2018	GB model	Identification of an infected host as an anchor point to reveal other compromised hosts through forensic graph-based algorithms. Reveal anomalies on rare paths through the detection of remote file execution
Host in danger? Detecting network intrusions from authentication logs [9]	2019	SSC (DT, RF, LiR, GNB, LaBi)	Hybrid anomaly detection perspective regarding the identification of LM techniques on hosts during the early stages of their threat exposure. They extracted 35 composite log-based features from the LANL dataset, which were classified under SSC ML algorithms
A machine learning approach for RDP-based lateral movement detection [10]	2019	SSC (DT, RF, LiR, GNB, LaBi)	Re-examination of the work presented in [9] under the prism of RDP-based LM
Analyzing system log based on machine learning model [12]	2020	SSC (SVM)—NN	Shallow and DNN ML classification analysis on Sysmon log samples
Uncovering lateral movement using authentication logs [11]	2021	SSC (DT, RF, LiR, GNB, LaBi)	The same authors of [9] and [10] revisit the subject of LM detection through the classification of LM-related authentication logs with shallow ML techniques
Anomaly detection for insider threats using unsupervised ensembles [14]	2021	UML (AE, IF, LODA, LOF, Unsupervised Ensembles)	Unsupervised ML detection of anomalies on user behavioral habits. Creation of Unsupervised ML ensembles to evaluate the performance of the anomaly detection scheme under various algorithmic combinations
Hopper: modeling and detecting lateral movement [22]	2021	GB model	Graph representation of user login activities to detect anomalies referring to LM incidents
LMTracker: lateral movement path detection based on heterogeneous graph embedding [23]	2022	GB model—NN	Presentation of the LMTracker custom LM identification algorithm. LMTracker is a mixture of LM paths representation via heterogeneous graphs construction and anomaly detection through graph-based neural networks algorithmic theory

munication, files creation and deletion, DNS queries, and more.

Windows Event Logging files, namely EVTX, is a dedicated for log information storing.xml format, that is exclusively used on MS Windows OS. As detailed in [26], although EVTX files obey the fundamental rules of the extensible markup language (XML) towards data representation, when it comes to file manipulation Microsoft differentiates. The EVTX format replaced its predecessor.evt introducing in parallel a great variety of new event properties, which can be used in the description of the stored events. The most commonly used properties with Sysmon's EVTX files are presented in Table 2. Sysmon is by default capable of simultaneously organizing and extracting the collected logs, through the windows event viewer (WEV) application, in a variety of files of multiple types, including EVTX,.xml, CSV and.txt formats respectively. Nevertheless, due to the core structure of the EVTX format files, as detailed below, all these export formats are cumbersome to manage and abundantly unsuitable for ML techniques.

Each EVTX file can contain millions of recorded event-related logs described through the multiple properties of Table 2 and set under two distinct XML sub-nodes, namely $< System >$ and $< EventData >$, respectively. Although the manipulation of such a hierarchically structured.xml file might seem a straightforward process, the reality is far beyond functional. Any attempt to extract the log-records from a EVTX repository in an equivalent CSV format ended in an incomplete outcome from the desired. Precisely, although WEV parsed each property of the $< System >$ node in a separate column, the information included in the $< EventData >$ sub-node was placed as a whole piece within the same cell of the CSV file. This particularity made the data incompatible for further preprocessing and analysis through ML experiments.

An example of this situation is given in Listing 1. Although all the properties of the $< System >$ sub-node (those within lines 2 to 15), were extracted as separated column-name in the CSV file, no property related to the $< EventData >$ tag (lines 18 to 22) was imported separately, but in a unified cell.

```
1  <System>
2     <Provider Name='Microsoft-Windows-Sysmon
        ' Guid='{5770385f-c22a-43e0-bf4c-06
        f5698ffbd9}'/>
3     <EventID>5</EventID>
4     <Version>3</Version>
5     <Level>4</Level>
6     <Task>5</Task>
7     <Opcode>0</Opcode>
8     <Keywords>0x8000000000000000</Keywords>
9     <TimeCreated SystemTime='2021-11-30
        T03:31:51.792896600Z'/>
10    <EventRecordID>14840</EventRecordID>
11    <Correlation/>
12    <Execution ProcessID='3064' ThreadID='
        3724'/>
13    <Channel>Microsoft-Windows-Sysmon/
        Operational</Channel>
14    <Computer>WIN-J23NIGGP1Q6.sysmon_set.
        local</Computer>
15    <Security UserID='S-1-5-18'/>
16 </System>
17 <EventData>
18    <Data Name='RuleName'>-</Data>
19    <Data Name='UtcTime'>2021-11-30 03:31:51
        .790</Data>
20    <Data Name='ProcessGuid'>{27532e6a-9a74
        -61a5-1601-000000001500}</Data>
21    <Data Name='ProcessId'>1144</Data>
22    <Data Name='Image'>C:\Windows\System32\
        conhost.exe</Data>
23 </EventData><System>
```

Listing 1 Example of Sysmon <EventID>5<EventID> properties (see EventID no. 5 in Table 2.)

3.2 ETCExp tool

To remediate this improper CSV export situation, we implemented a tool coined ETCExp. The essence of ETCExp is to provide a lightweight, portable, easily configurable and above all OS-independent command line (of IDE executable) tool that helps incident response teams and researchers to parse and make massive EVTX log files compatible to be implemented into ML algorithms. The tool can process large EVTX files very quickly; for instance, the conversion of a 1.41 GB (2.7 M samples) corpus takes around 30 min. The tool is publicly available as open-source in [27].

Briefly, ETCExp has dependencies in two main Python libraries, namely ElementTree and Pandas. It allows the following operations to be performed:

- Memory mapping of massive Sysmon EVTX log files extracted with WEV in.xml form.
- Parsing of each EventID's properties based on preconfigured filtering parameters, and transformation into CSV column-headers as a distinct EventID.
- Provision of the existence of duplicate column-headers and implementation of each property's value based on an incremental index number.
- Parallel examination of each EventID's implementation process per index for empty cells and placement of the NaN value.
- Creation of Python Dataframe with the total of the parsed logs as output.
- Transformation of the Python's Dataframe into a CSV file.

From an OS version's perspective, ETCExp can be executed on any mainstream platform, namely Windows 11,

Table 2 Description of Sysmon's logs EVTX format properties

Property	Description	Property	Description
Computer	The computer's identification name on which the LM event occurred or through which the event was forwarded	*ProcessorID*	Identification number of the Processor that processed the event
Correlation_ID	Specifies the activity which is involved in the recorded event	*Processor_Time*	The elapsed time of the user within which executed instructions, in CPU time ticks. CPU time ticks determine how fast an instruction is executed within the processor
Date_and_Time	The date and time the event took place	*Relative_Correlation_ID*	Identifies related activity to processes that triggered the event
DestinationPortName	The port number where the connection was received	*SessionID*	Identification number of the terminal's session in which the event happened
EventID	The distinct numerical identifier, of the general predefined by Microsoft category, which describes each event-related log	*Source*	The name of the software, system component or driver that triggered the creation and storage of the event-related log
EventRecordID	The dedicated number assigned to each record at the time of being logged	*SourceIsIpv6*	The IPv6 of the station from which a connection was initiated
ExecutionProcessID	The unique identifier that Sysmon assigns to each process at the starting point of its execution	*SystemTime*	The dedicated time that the system captured the generation of the related *EventID*. Initially the OS captures the time in *UTC* but each SIEM tool, like Sysmon, converts it to the local time of each user. In the context of this paper, the time is *UTC+2*
Initiated	Boolean value that indicates the initiation or not of network connections	*Task_Category*	Denotes sub-categories of the logged event
Level	The escalation of the severity of the event. It includes six different categories (namely Information, Warning, Error, Critical, Success Audit, and Failure Audit)	*ThreadID*	Identification number of the thread that triggered the event
Log	The log's name	*User*	The user's identification name on behalf of which the LM event occurred or through which the event was forwarded
Operational_Code	A numeric value which identifies the activity which is involved in the recorded event	*User_Time*	The elapsed time of the user within which executed instructions, in CPU time units
ProcessID	Identification number of the process that triggered the event		

Table 3 Overview of the LMD-2022 dataset [24]

LMD-2022 subsets	# Sysmon logs	Network traffic/LM techniques
Normal	$\approx 80K$	Legitimate network traffic (LNT)
NormalVsMalicious01	$\approx 290K$	LNT, EoRS (ms17-010, EternalBlue, Bluekeep, WannaCry)
NormalVsMalicious02	$\approx 415K$	LNT, EoHT (PtH, PtT, GT, ST [via Mimikatz], LaZagne Project)
FullSet	$\approx 870K$	LNT, EoRS, EoHT

macOS Ventana v13.0 and Ubuntu v22.04 LTS. With basic prerequisite the installation of Python's dependencies and the import of the corresponding packages, the core functions of ETCExp are executed revealing their potentials, as presented in Algorithm 1.

Precisely, the installation of Python's dependencies is followed by the import of Pandas and xml.etree.ElementTree packages. The core functionality of ETCExp can be summarized in the notion of parsing each EventID's properties through the ElementTree's conditional statements, as delineated in Algorithm 1, lines 3 to 28. In further detail, ElementTree is a core Python library dedicated to the representation of XML files, as Sysmon's EVTX format denotes, in a tree-based hierarchical structure. According to its publicly available documentation, the library permits two different levels of parsing interaction: the first on the whole XML document on an ElementTree level, and the second on the numerous sub-nodes of an XML file on an Element level specifically.

Regarding the parsing of the extracted in.xml format EVTX Sysmon files, which it is initiated after the whole document is buffered on system's memory via the ElementTree's *toString()* function, as presented in Algorithm 1, line 15. The string variable is reprocessed through the *fromString()* function to be finally stored in the *root* variable and is set as input for the rest of the parsing process, as presented in line 16 of Algorithm 1. The collected.xml sub-node entries of the *root* object are parsed through a combination of *if, elseIf and forIf* loops, based on predefined $< EventID >$ filtering tag values. The output is finally added to a Python's key-value pairs dictionary through the *addValue()* function (line 20 in Algorithm 1). It should be noted that the *addValue()* function is not native to ElementTree, but custom-made to fit the needs ETCexp. The finally created *dict()* object is manipulated through Pandas and extracted in the desired CSV format. For further details regarding the structure and operation of ETCExp, the reader is referred to the corresponding GitHub repository in [27].

Algorithm 1 ETCExp's algorithm

```
Require: python $setup.py$ install
Require: pip install $pandas$ library
Require: import $mmap$, $xml.etree.ElementTree$
Require: input() $data$, $Sysmon$ files in $EVTX$ format extracted
    through $WEV$ as $.xml$ file
Require: dict()
Require: Function def $add\_value(dict\_obj, key, value = None)$ :
    if $value \neq None$ then
        $dict\_obj[key] \leftarrow ['0', 'testValue', value]$
3:  else if $value == None$ then
        $dict\_obj[key] \leftarrow ['0', 'NaN']$
    else if $isinstance(dict\_obj[key], list)$ then
6:      if $value \neq None$ then
            $dict\_obj[key] \leftarrow append(value)$
        else if $value == None$ then
9:          $dict\_obj[key] \leftarrow append(NaN)$
        end if
    else if $value \neq None$ then
12:     $dict\_obj[key] \leftarrow [dict\_obj[key], value]$
    end if
    $xtree \leftarrow ElementTree[parse()](.xml\_file)$
15: $xmlstr \leftarrow ElementTree[toString()](xtree)$
    $root \leftarrow ElementTree[fromString()](xmlstr)$
    for $child \in root$ do
18:     for $sub\_child \in child$ do
            if sub_child.tag == tag.value then
                $add\_value(dict()$
21:             $(sub\_child.attrib).items()$
                $sub\_child.text()$
            end if
24:     end for
    end for
    $DataFrame \leftarrow pandas(dict\_obj)$
27: $CSVfile \leftarrow pandas(DataFrame)$
    $return$
```

3.3 Proof of concept

The soundness of ETCExp was assessed against a real, contemporary EVTX corpus, namely LM dataset (LMD) [28]. This will serve a dual goal: first, it allows us to verify the appropriateness of the tool for further use by the research community, and second, using the derived dataset, estimate the LM detection capacity of several ML techniques through a well-defined methodology. Regarding the second goal, to our knowledge, and based on the discussion in Sect. 2, the current work is the first to explore this potential through the use of logs created by Sysmon.

To the best of our knowledge, the LMD dataset [28] is currently the only benchmark corpus comprising Sysmon logs. The 2022 LMD version (LMD-2022) incorporates normal and malicious traffic logs originated from the execution of nine state-of-the-art LM techniques, including four variants of the so-called Exploitation of Remote Services LM methodology and five equivalents credential exploitation techniques. The attacks were recorded under the MS Windows Domain testbed presented in [24]. Specifically, the nine assaults range from the execution of the legacy Exploitation of Remote Services techniques via the ms17-010, EternalBlue and Bluekeep Windows vulnerabilities, to the more advanced deployment of the WannaCry malware and more elevated ones, including "Pass the Hash" (PtH), "Pass the Ticket" (PtT), "Golden Ticket" (GT), "Silver Ticket" (ST) and credential exploitation with LaZagne Project tool.

As shown in Table 3, the LMD-2022 corpus comprises four subsets that were thoroughly presented in [24], namely Normal, NormalVsMalicious01, NormalVsMalicious02, and FullSet. Specifically, the Normal subset incorporates logs related to legitimate network traffic that were collected prior and during the execution of nine executed LM techniques, namely "exploitation of remote services (EoRS)" (four variants of EoRS techniques) and "exploitation of hashing techniques (EoHT)" (five variants of EoHT techniques). The variants of the nine executed LM techniques per subset are presented in the rightmost column of Table 3. Precisely, NormalVsMalicious01 set of logs encloses the traffic that was captured before, during and upon termination of the "Exploitation of Remote Services (ERS)" attacks category, while NormalVsMalicious02 comprises logs collected during the execution of the five aforesaid distinct credentials exploitation techniques. Both, NormalVsMalicious01 and NormalVsMalicious02 are mixed with normal traffic logs respectively. Finally, FullSet is the fusion of the three aforesaid distinct subsets.

For the needs of this work, the LMD-2022 logset was enhanced with both normal and malicious traffic collected from various personal and virtual machine (VM) computing stations. Precisely, the already existing nine LM techniques were re-executed multiple times and populated with six more up-to-date state-of-the-art LM techniques, which correspond to common vulnerabilities and exposures (CVEs) IDs issued from 2020 until today. As shown in Table 4, the added attacks (those having an asterisk affixed) include Log4Shell, Follina, Windows Spooler Privilege Escalation, SMBGhost, SMBleed, and Zerologon, each of them executed multiple times. The resulted LMD-2023 dataset, which is used in the context of this work, comprises a full set of 1,752,890 log samples (EventIDs). LMD-2023 is offered in both CSV and Sysmon's generic EVTX (raw.xml data) formats. The CSV file contains 93 features, and it was produced through the use of the ETCExp tool described in Sect. 3. The 16 distinct EventIDs (out of the total 27 presented in Sysmon's manual [26]) identified by the ETCExp when parsing the corresponding EVTX file are included in Table 5. Note that the rest 11 Sysmon EventIDs, such as EventID_6 (Driver loaded), Event_8 (CreateRemoteThread) etc., are not present in LMD-2023 due to the nature of the implemented EoRS and EoHT techniques.

3.4 Dataset labeling

When it comes to supervised ML techniques, these are defined by the use of labeled data samples during the training process of each algorithm. This, however, requires the execution of a labeling process on the dataset. For LMD-2023, three classes were defined having the labels presented in Table 6, namely Normal, EoRS and EoHT. The latter two classes represent the 15 LM techniques contained in Table 4. Particularly, the correspondence between these two classes and the matching LM method is given in the rightmost column of the same table.

For categorizing each of the ≈1.75 M samples of LMD-2023 into one of the three above-mentioned classes, the various criteria available in the EDR policy presented in [24] were imported as labelling filtering rules into an independent Python script, that is publicly available as open-source in [27]. This EDR policy comprises a collection of impactful LM rules for determining the optimal initialization features of Sysmon regarding the EoRS and EoHT categories. Precisely, the rules were derived from extensive experimentation and thorough empirical observation upon various LM techniques. All the rules were also applicable as custom rules within a provided Sysmon's config.xml configuration file.

Furthermore, as a proof of concept for the aforementioned rule-based policy's efficiency, we implemented PeX tool, which is publicly available in [25]. The tool caters for the analysis of voluminous Sysmon logs through the enumeration and dedicated filtering of Windows Event Logger and Sysmon EVTX file entries, aiming at revealing the existence of possible LM traces over small office home office (SOHO) networks. PeX not only automatically analyzes and identifies any kind of logging activity captured by Sysmon, either normal or malicious, but also serves as a first step for the incident response teams towards the identification of LM events. For more information on the employed rules, the reader is referred to §6 and Appendices A.1 and A.2 in [24].

The results of the labeling process, following a manual verification process, are given in Table 6; obviously the dataset is highly unbalanced, especially regarding the EoHT class.

4 Methodology

4.1 Feature selection

As already pointed out, the current work relies on a set of classification features with the aim to detect LM events. Feature selection is a fundamental step when it comes to the manipulation of voluminous datasets via ML techniques. For this reason, it was conducted under various criteria related to common expertise in this field. Above all else, feature selection should be based on the notion that each feature must enclose adequate numerical, Boolean, or textual information, which through the right preprocessing, could contribute with positive impact to the learning process. The process of reducing the initially extracted 93 features of LMD-2023 CSV file with the ETCExp tool into a smaller but of high importance subset was rather straightforward. Precisely, features like *Name*, *Guid*, *Opcode*, *Keywords*, *Correlation*, *Channel*, *State*, *Version*, *StartFunction* and *ID* were dropped from the original set of features in [25] due to not bearing any useful information in assisting the detection of LM techniques. That is, such features carry the same value (or values) across all samples, as presented in Table 7:

The values of the remaining 83 features were scrutinized based on the acquired insights from the studied literature in Sect. 2. Moreover, based on our expertise in the field of LM

Table 4 LM techniques included in the LMD family of datasets. The star exhibitor denotes attacks included in the LMD-2023 version

LM technique	CVE ID(s)	LM Class
ms17-010	CVE-2017-0148	EoRS
EternalBlue	CVE-2017-0144	EoRS
Bluekeep	CVE-2019-0708	EoRS
WannaCry	CVE-2017-0143, CVE-2017-0145, CVE-2017-0146	EoRS
Mimikatz (EoHT)	CVE-2021-36934	EoHT
LaZagne Project	CVE-2021-40444	EoHT
Log4Shell*	CVE-2020-1472, CVE-2021-44228	EoRS
Follina*	CVE-2022-30190	EoRS
Windows Spooler Privilege Escalation*	CVE-2022-29104	EoRS
SMBGhost*	CVE-2020-0796	EoRS
SMBleed*	CVE-2020-1206	EoRS
Zerologon*	CVE-2020-1472	EoRS

Table 5 Sysmon's EventIDs included in the log files of the LMD-2023 Dataset. The star exhibitor denotes normal traffic exclusively

No.	EventID	Description
1	EventID 1	Process creation
2	EventID 2	A process changed a file creation time
3	EventID 3	Network connection
4	EventID 4	Sysmon service state changed
5	EventID 5	Process terminated
6	EventID 7	Image loaded
7	EventID 10	ProcessAccess
8	EventID 11	FileCreate
9	EventID 12	RegistryEvent (Object create and delete)
10	EventID 13	RegistryEvent (Value Set)
11	EventID 16	ServiceConfigurationChange
12	EventID 17	PipeEvent (Pipe Created)
13	EventID 18	PipeEvent (Pipe Connected)
14	EventID 22	DNSEvent (DNS query)
15	EventID 23	FileDelete (File Delete archived)*
16	EventID 255	EventID 255: Error*

Table 6 Structure of the LMD-2023 labeled dataset

CN	CL	# samples	% over LMD-2023
Normal	0	1,611,637	≈92%
EoRS	1	110,746	≈6%
EoHT	2	30,507	≈2%

CN, class name; CL, class label

techniques execution, we concluded with a smaller set of generic features, totally bonded to the characteristics of the attacks presented in Table 4. Altogether, feature selection was performed contingently on the following key conditions:

– Each sample must include adequately representative to each LM technique numerical, Boolean, or textual information and be as independent as possible of the testbed settings on which the attacks were executed; this way an attacker cannot trivially bypass the ML algorithm by modifying the features. On the contrary to the aforesaid, generally accepted by the bibliography, notion of features independence to local settings, features like *EventRecordID*, *ExecutionProcessID*, *ProcessID* and *Computer* were included in the finally selected features of Table 8. This choice is substantiated by the study of the structural characteristics of each LM technique and more precisely by the fact that the cornerstone of each attacker's LM behavior is to gain remote or physical access within a secure location, remain stealth for as long as possible and through the simulation of processes and active applications of the compromised computing structure incrementally and laterally reach its ultimate target. Therefore, incorporating features representative to testbed's stations processes and users not only is imposed by the samples structure, but also contributes positively to ML models learning and predictions, as it will is detailed in Sect. 4.2.

– Regarding the selection of the EventID, *Initiated*, *SourceIpv6* and *DestinationPortName* features presented in Table 8, we relied either on the relevant literature (how often the feature becomes exploitable in similar analyses), or on empirical and experimental observations (which is the actual information a feature carries for the detection of each of the three classes, namely Normal, EoRS and EoHT.)
– As it concerns the finally selected *SystemTime* feature, although a part of the studied bibliography suggests not using features of time or flow oriented series, we argue that within the overall context of the execution of LM techniques, System's time in the log-based form of Sysmon samples offers a self-sufficient source of information. The experiments of Sect. 5 evaluate the feature's ability to contribute in combination and incrementally to the rest of the selected features of Table 8, increasing the relevant prediction scores (especially AUC and $F1$) of ML models. Despite the usefulness of time-oriented features, these samples come in immutable form "2022-06-25T19:30:32.4333805Z" which does require feature engineering preprocessing for being transformed to useful numerical equivalent data. That is, we replaced *SystemTime* attribute with seven equivalent time derivatives features, related exclusively to each event's *year*, *month*, *week*, *day*, *hour*, *minute* and *day within the week*. For instance, "2022-06-25T19:30:32.4333805Z" is manipulated with Panda's *to_datetime()* function, dividing it into two different elements, namely "2022-06-25T" and "19:30:32.4333805Z", respectively. Through the same function, the two special characters "T" and "Z" were also eliminated. Next, the format of the timestamps was altered with the *astype("datetime64[s]")*, and was finally collected per gender into separate columns to produce the seven aforesaid time features given in Table 8.
– Overall, we resulted with nine features. The first eight are contained in the top part of Table 8. The remaining one, namely *SystemTime*, yielded seven features, namely *SystemTime_year*, *SystemTime_month*, *SystemTime_week*, *SystemTime_day*, *SystemTime_hour*, *SystemTime_minute* and *SystemTime_day_of_week*, contained in the bottom part of the same table.

Based on the above-mentioned set of criteria, and as summarized in Table 8, the final feature subset that is used as the base for the preprocessing procedure of Sect. 4.2 and as input in the shallow and Deep NN experiments of Sect. 5 comprises 15 features. As it concerns the numerical representation of Normal traffic against the equivalent Malicious, the dataset is imbalanced.

4.2 Data preprocessing

In the context of this work, data preprocessing refers to the sequence of tasks that follows the transformation of Sysmon logs into compatible with ML algorithms CSV formed files. Specifically, data preprocessing is involved with the encoding, normalization, and scaling of the raw data so that it may be easily parsed by the machine. One-hot encoding (OHE) and min-max scaler (Min-Max) are the commonest methods for manipulating categorical and alphanumeric data, respectively. Precisely, OHE algorithm is designed to deal with categorical data, which are represented as variables that store labels instead of numerical data. Categorical data may either be nominal (grouped variables based on a common specific characteristic) or ordinal (variables which enclose data with some sort of natural relationship, through which may be ordered). Despite the applicability of OHE on ML experiments, there are several shallow classifiers, such as Decision Tree (DT), which according to the specific characteristics of each dataset may be trained directly though categories. The rationale behind the final selection of the OHE preprocessing algorithm for the manipulation of categorical features hinges on the fact that the majority of the available shallow classification techniques require all data to be processed to scientifically acceptable numerical format. OHE deals with categorical values preprocessing by adding for each unique categorical value a new binary equivalent. As many are the distinct categorical values within a feature, OHE will produce an equivalent number of new binary derived features.

As it concerns the scaling of numerical data, the two most prevalent methods are Normalization and Standardization. Regarding the former, each sample is scaled separately within a range of 0-1 that is related to the floating-point values which perform better in terms of Precision and Recall metrics. Standardization refers to the process of scaling each sample's value separately through the use of mean subtraction and division with each sample's standard deviation (STD). Although Standardization is rather an effective scaling method, within the context of this study, the method of Normalization was adopted as the most equivalent to fulfill the preprocessing of the numerical features, as recapitulated in Table 8. Particularly, Normalization techniques fit adequately to the distribution of numerical features of the LMD-2023 dataset that is limited within 0 and 1 values. For both OHE and Normalization techniques, the sklearn version 1.0.1 Preprocessing package was utilized, specifically the OneHotEncoder() and MinMaxScaler() algorithms, respectively. It is noteworthy that the LMD-2023 dataset was

Table 7 Excluded features from LMD, because they carry the same price along the total of the samples. The second column presents the value of each feature

No.	Excl. feature	Value
1	Name	Microsoft-Windows-Sysmon
2	Guid	{5770385f-c22a-43e0-bf4c-06f5698ffbd9}
3	Opcode	0
4	Keywords	0x8000000000000000
5	Correlation	0
6	Channel	Microsoft-Windows-Sysmon/Operational
7	State	Started or 0
8	Version	13,24 or 0
9	StartFunction	0
10	ID	GetConfigurationOptions or 0

Table 8 The 15 selected features and the data preprocessing method applied to each one

Sysmon's feature	Preprocessing method
Computer (CompSTA)	OHE
DestinationPortName (DstPortName)	OHE
EventID	OHE
EventRecordID (EventRecID)	MinMax
Execution ProcessID (ExecProcessID)	MinMax
Initiated (Init)	OHE
ProcessId	MinMax
SourceIsIpv6 (SrcIpv6)	OHE
SystemTime_year (SysTimeYear)	OHE
SystemTime_month (SysTimeMonth)	OHE
SystemTime_week (SysTimeWeek)	OHE
SystemTime_day (SysTimeDay)	MinMax
SystemTime_hour (SysTimeHour)	MinMax
SystemTime_minute (SysTimeMinute)	MinMax
SystemTime_day_of_week (SysTimeDoW)	OHE

The abbreviated titles of the presented features in parentheses are provided for presentation and space economy reasons, for the feature importance stacked diagram in Fig. 1

preprocessed in its original length of 1,752,590 samples, without applying any sampling techniques.

The resulting CSV file was filtered via the Pandas Python library for undefinable values that could cause the ML algorithms to produce errors. That is, the undefined per algorithm values included "NaN", "Null", diminutive floating-point values, any value expressed in the form of scientific notation (such as 3.456e11) and dash ('–') values related to the 'DestinationPortName' feature. The corresponding rows of the aforementioned values, were counted and found to represent 3.5% or 66,150 samples of the dataset and for that reason they were dropped along with their corresponding rows. Empty cells were also filled with 0 via Python's *fillna()* function.

The resulted labelled and preprocessed CSV file was exploited in its original imbalanced form, regarding the numerical representation of each Label within the dataset. Interestingly, the EoHT samples were only 30,507 or 1.7% compared to the rest of LMD, a fact that is expected to stretch the performance of the majority of the classifiers.

4.3 Feature importance

As already mentioned in Sect. 4.1, the selection of the 15 features recapitulated in Table 8 stems from the intense study of the presented literature in Sect. 2 and our own expertise in this field. The majority of the eight selected features, including the seven others exported from the *SystemTime* equivalent, were searched and found to include adequate numerical, Boolean or textual information related to LM. However, for precautionary reasons and for revealing any potential negative impact that may be imposed on the experiments with shallow classifiers in Sect. 5.1, we additionally conducted two different feature importance techniques.

Specifically, when it comes to ML experiments, the final metrics of the performed experiments are highly prone to noise originating from non-equivalent to the examined problem's features. If this parameter is ignored, then the results from the majority ML repetitive applications will be biased, possibly leading to wrong results. Feature importance techniques present in a straightforward manner the "importance" score, demonstrating how each feature contribute vis-à-vis the total of the dataset.

In this work, two independent importance analysis techniques were conducted on the already preprocessed with OHE and MinMax algorithms LMD-2023 CSV file: (a) examination of the model's coefficients, and (b) examination of the LMD's Principal Component Analysis (PCA). The analysis used the 100% of the stratified data from each feature set. Precisely, each from the LogisticRegression(), and PCA() sklearn's algorithms were trained with the 70% of the stratified sample and tested with the rest 25% subset of the stratified data.

Regarding the examination of the model's coefficients, the method was applied on the dataset via the LogisticRegression() algorithm, leading to an equation in which coefficients (importances) are assigned to each input value. Put simply, large (negative or positive) coefficient numbers, impose some influence on the prediction. Contrariwise, if the coefficient is zero, it does not have any impact on the prediction. After the model's fitting, the coefficients are stored in the *coef_* variable.

As depicted in Fig. 1, the analysis of the LMD-2023 preprocessed features revealed that 49 of the total 91 features offer the most information. Specifically, *ProcessId* was assigned the best coefficient that almost reached 1. Six more, namely *SysTimeMonth_6*, *CompSTA1*, *SysTimeWeek_25*, *SysTimeYear_2022*, *DstPortName_netbios-dgm* and *init_True* revealed an importance within the range of 0.6 to 0.8. The rest of the 42 positive coefficients range from almost 0 to 0.5.

In turn, as also illustrated in Fig. 1, PCA was implemented to reveal corresponding results to feature importance analysis via Coefficients and LoR algorithm. PCA is placed in the unsupervised learning techniques specifically designed to deal with high-dimensional datasets. Typically, this technique is used for the reduction of data dimensionality, prior to the creation of an ML model. This is done due to its robustness in overfitting and data loss, and for those reasons it is ideal for estimating the most significant features of a dataset.

Overall, with reference to Fig. 1, it is very promising that the initial decision upon the selected set of features in Table 8 were supported by feature importance analysis to enclose adequate amount of information.

5 Experiments

The current section details the conducted experiments and discusses the derived results. As stated in Sect. 4.2, only the most significant preprocessing alterations were done on the features of Table 8, in an effort to achieve the best possible generalization. The most discussed in bibliography ML techniques were implemented, avoiding introducing any hyperparameters optimization and dimensionality reduction techniques. Moreover, no custom (time or any other vector related) feature was considered. With reference to the methodology, the following points worth to be mentioned:

- All the ML algorithms presented in Table 9, along with their initialization hyperparameters of Table 10, were chosen based on reproducibility criteria and upon the notion to be freely available for implementation in popular ML libraries.
- All the algorithms included in each experimental set were utilized without any alteration regarding the settings of Table 10). This was done for maintaining a standard base of hyperparameters and for reasons of generalization.
- With reference to Table 6, due to the highly imbalanced nature of LMD-2023, the focus is on two metrics, namely Area Under the Curve (AUC) and $F1$. AUC is the ML measure of separability along the various labels of a multiclass ML model, and its value is extracted from the receiver operating characteristic (ROC) curve. It is a probability plot that represents the graphical representation of the true-positive Rates (TPR) against the false-positive Rates (FPR), under various predefined thresholds. On the other hand, AUC demonstrates the representative value of the aggregated performance of binary classification algorithms. The more the AUC value is closer to 1 the best is the ML model to distinguish Normal vs Malicious classes. Due to the fact that the three defined classes of Table 6 consist a multiclass ML problem, one-vs-all *LabelBinarizer - LaBi* schema had to be implemented for extending the binary classification schema to the multiclass case. *LaBi* is included in the preprocessing package of the Sklearn Python library, and utilizes as input the test (or $\approx$ 25% of the evaluated samples) part of the LMD dataset along with the equivalent predicted fractions of each ML algorithm. Regarding the $F1$ score, it is the ML metric that is proposed for imbalanced datasets in the place of Accuracy. $F1$ is calculated as the harmonic-mean of the two aforementioned values, based on the formula $F1 = 2 * ((\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall}))$.
- Because of the imbalanced numerical characteristics of the LMD dataset the stratified k-fold Cross Validation, with a $k = 10$; this prevents overfitting. Each fold had 1,314,668 and 438,223 samples (100% of the total LMD-2023 dataset) for the training and testing sets, respectively.
- For avoiding overfitting, various hands-on regularization hyperparameters attempts were also conducted, including *tol*, *early_stopping*, *max_depth*, *reg_alpha*, *reg_lambda*, *ccp_alpha*, etc.

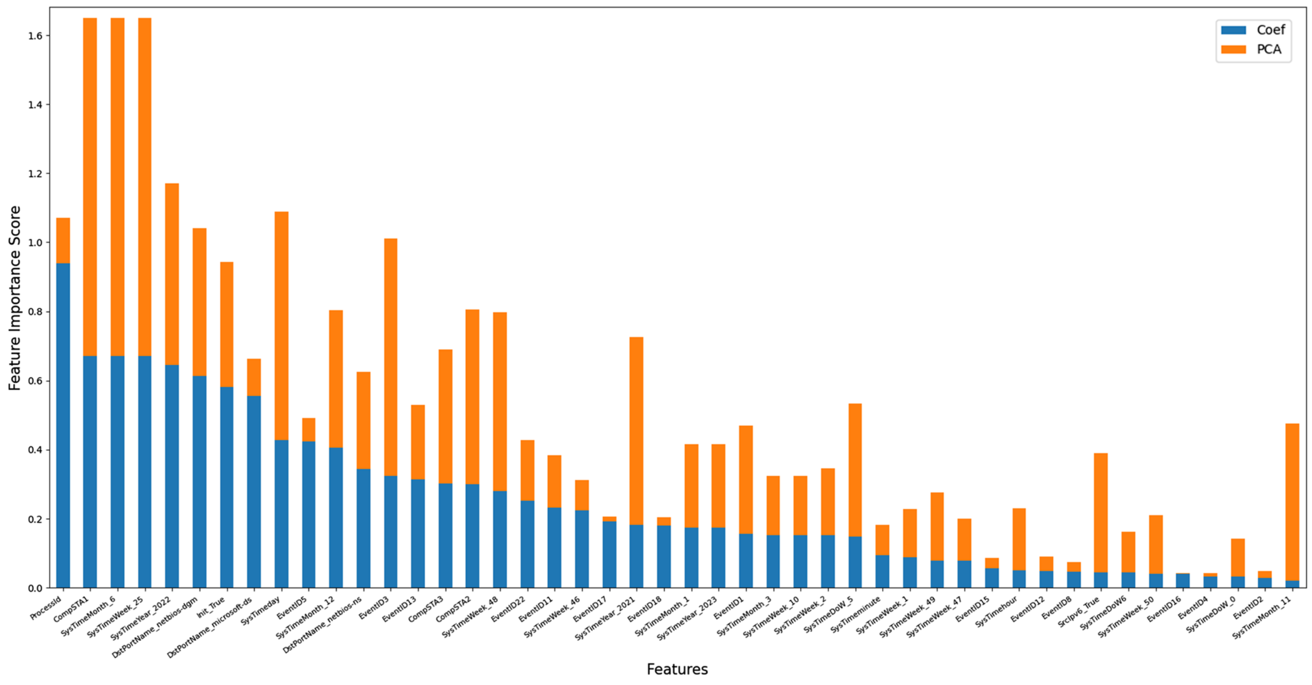


Fig. 1 Feature importance through LMD-2023 sample's coefficients and PCA for the total of feature sets. All the insignificant features <0.05 were removed. The names of the analyzed features are abbreviated as given in Table 8

Table 9 Shallow and DNN algorithms employed for evaluation purposes

Algorithm	Category	Genre
LoR	SC	Probabilistic
SGDC	SC	ST-based
KNN	SC	Deterministic
NB	SC	Probabilistic
LSVC	SC	Linear-based
LGBM	SC	Tree-based
DT	SC	Tree-based
RF	SC	Tree-based
ET	SC	Tree-based
CB	SC	Tree-based
Bagging	SC	Tree-based
MLP	DNN	Multilayer FF ANN
CNN	DNN	ANN
LSTM	DNN	RNN
RNN	DNN	Multilayer FF ANN
Autoencoders	DNN	ANN

– The key aim is to evaluate whether the cherry-picked 15 features of Table 8 can yield acceptable detection rates in the context of an IDS. Therefore, no hyperparameter tuning methods, such as Python's *Optuna* or *Grid search* were followed.

5.1 Shallow classifiers

Shallow classification on the labeled LMD-2023 dataset was conducted against a range of base estimators and ensemble meta-estimators, i.e., the topmost 11 classifiers of Table 9. The decision to use such a broad repertoire of algorithms was done for comparison reasons and because, with reference to Sect. 5.1.1, each technique enclosed special functional characteristics with potentials on the creation of a log-based IDS. The models were built on an MS Windows 10 Home Edition (v. 21H2, OS Build: 19044.2486) machine, with Intel(R) Core(TM) i7-9750 H CPU @ 2.60GHz 2.59 GHz CPU, 64.0 GB RAM and NVIDIA GeForce GTX 1660 Ti GPU. The experiments were run solely on the base-machine's CPU and RAM, without GPU acceleration. All the relevant scripts were developed on Python 3.9.2 via Sklearn 1.0.1, Pandas 1.4.4, Numpy 1.21.2, Seaborn 0.11.2 and matplotlib 3.4.3 libraries and packages. LightGBM and CatBoost algorithms were implemented via the homonym packages.

5.1.1 Configuration of hyperparameters

The analysis centers on five distinct experimental classification categories, namely Probabilistic, Stochastic-based, Deterministic, Linear and Tree-based algorithms, as presented in Table 9. During the preparation and parameterizing of each algorithm, our goal is not to achieve optimal results, but to propose a generalized methodology for supervised learning in LM intrusion detection.

Probabilistic algorithms are dependent on the mathematical concept of probability to train their models and calibrate their demonstrated predictive analysis metrics. In the context of this work, the expanded version of the optimizer Stochastic Average Gradient (sag), namely *Saga*, was utilized as the solver in assisting the Logistic Regression classification procedure. *Saga* solver is an elevated extension of *sag*, suitable for fast training upon large multiclass datasets. For that reason, and for handling the multinomial loss, the *multi_class* variable was set from *auto* to *multinomial*. The *max_iter* variable, was set to 1K maximum iterations until the solver reaches its ultimate convergence. Moreover, the algorithm's stopping criteria were optimized for terminating the learning process when the *tol* variable of tolerance is not improved by at least 1e−3 within two consecutive iterations of the dataset's training. Additionally, the *Saga* solver supports all three available regularization penalties, namely *L1*, *L2* and *elasticnet*. For the needs of the present work, the *elasticnet* was chosen because it adds the effects of *L1* and *L2* penalty hyperparameters into a finally aggregated value. Both penalty values are among the most effective hyperparameters for mitigating overfitting effects in several ML algorithms, especially the ones included in sklearn Python library. L1 is called *lasso* regression, and it adds the absolute value of each coefficients magnitude. L2 is called *ridge* regression, and it adds the coefficients squared magnitude. Both parameters use as their core values coefficients, therefore they are applied in feature selection too as it is described in Sect. 4.3.

NB is another parameterization independent probabilistic algorithm based on the Bayes theorem of calculating conditional probabilities through mathematical formulas. As a result, NB incorporates the "naive" assumption that each pair from the selected features is independent of any condition regarding its corresponding value in the class variable. Consequently, and according to the assumptions each time the algorithm makes upon each distribution, it comes in four different variant classifiers, namely Gaussian (GNB), Multinomial (MNB), Complement (CNB) and Bernoulli (BNB). In this work, we focused on the CNB, which is the extended version of the standard MNB that is exclusively calibrated for imbalanced datasets, like the utilized LMD-2023.

On the other hand, Stochastic-based models are highly characterized by their tendency to reveal randomness and uncertainty on outcomes related to varying inputs. Precisely, the SGDClassifier expands on known linear classification ML models, namely SVM, LSVC, etc. through the training of each model via Stochastic Gradient Descent. Regularization of the loss of trained models is accomplished through the implementation of the same sklearn penalty values, namely *L1*, *L2* and *elasticnet*. In our study, due to the highly imbalanced nature of LMD-2023, the *m_huber* loss function was set to handle the tolerance of the outliers in each algorithm's execution. To avoid overfitting, the *learning_rate* was set to *optimal* and the *early_stopping* parameter was enabled. With respect to regularization values, *max_iter*, *tol* and *penalty* were set to 5e−3, 1e−8 and *elasticnet* respectively, while the *alpha* variable was assigned with the 1e−3 value to make the regularization effects stricter.

The deterministic KNN algorithm was configured (by default) with the *auto* for choosing the best algorithm while computing the distance between each neighbor, the *minkowski* criterion as the metric for calculating each distance, and the *uniform* method of weighting each distance equally during predictions. To improve the algorithm's fitting on the training and testing samples, the *leaf_size* and *n_neighbors* values were set to 50 and 10, respectively.

As it concerns the LinearSVC model, the most critical to the training process and handling of loss hyperparameters, namely *penalty* and *loss*, were set (by default) to *l2* and *squared_hinge* respectively. The only regularization amendments were made to the *max_iter* and *C* hyperparameters, as they were set to 30K and 1.5 each. Similarly to the two probabilistic models (Log.Reg. - CNB) and the SGDClassifier, the One-vs-Rest (OVR) heuristic binary classification over multi-class problems ML schema was also implemented in LSVC.

Six variants of the Tree-based category are included in Table 10, of whom the four belong to the sklearn classification library, namely DT, RF, ET and Bagging ensemble meta-estimator, while LGBM and CB are introduced as independent libraries. All the algorithms presented under this category are non-parametric, relying on function approximation to train as close to the tested samples as possible and cover the maximum range of different data shapes. The base algorithm of the aforesaid category is DT. It requires little data preprocessing effort (such as removing *null* and *NaN* values) and can be used for both numerical, categorical, or mixed multiclass datasets. Despite its simplicity, in the case of large datasets with multiple features, DT may be expanded over complex trees, leading to overfitting. To avoid such problems, we set the minimum number of the training inputs per leaf to 8 for ignoring any leaf that has fewer samples. Another critical parameter for countering tree models overfitting is *max_depth* that was set to 20 to calibrate the length of the longest path from the root of the tree to the last leaf. Accordingly, *max_leaf_nodes* was set to 100 and the *entropy* criterion was selected to test the quality of splits along leaf nodes. Moreover, the "Minimal Cost-Complexity" pruning *ccp_alpha* parameter was set to 1e−3 for removing the leaf-nodes that use features with low importance. In this way, the complexity of the model is minimized and its predictive power is elevated against overfitting. Finally, the *max_features* parameter, which is responsible for the identification of the optimal number of features during the split of the samples into branches and leaves, was set to square-root.

Regarding RF and ET, the same values per parameter as with DT were set for testing the algorithm's ability to generalize on the dataset. Nevertheless, as observed from Table 10, the classifier demanded more estimators, *max_depth* and *max_leaf_nodes* to generalize well, namely 1000, 300, and 650, respectively. The *ccp_alpha* pruning parameter was also set to an increased value, 1e−2.

LightGBM was configured with the default value regarding the boosting parameter type, namely Gradient Boosting Decision Tree or *boosting_type* = *"gdbt"*. LightGBM introduces a leaf-wise approach regarding the expansion of the trees, which contradicts the traditional Gradient Boosting methods of adapting depth-wise tree growth, such as the XGBoosting algorithm. The aforesaid approach permits LGBM to converge faster upon expanded datasets (>1.5 M samples), however with high probability of overfitting; therefore, the use of custom regularization hyperparameters was a necessity. First off, the maximum depth of the trees *max_depth* was set to 20, as in DT and RF. The methodology of adapting a specific depth, in conjunction with a proper initialization of the regularization hyperparameters across all leafs, will lead to the creation of a model which generalizes well across multiple sets of data. Therefore, the number of leaves was set to 20. For LGBM, it is recommended that the maximum number of bins (*max_bin*) should be kept to the smallest possible for achieving balance between the algorithm's speed and accuracy. Thus, the *max_bin* parameter was set to 20. The same policy was kept with the *min_child_samples*, *min_data_in_bin*, and *min_data_in_bin* hyperparameters that were kept small in accordance to the *max_bin* value. The remaining LGBM's hyperparameters comprise the *n_jobs*, the *n_estimators*, the *reg_alpha* and the *reg_lambda*. The first is related to the number of threads that will be used in parallel during the training of the model which is recommended to match the number of the physical processor's cores; eight for this work. The value of 100 estimators was assigned to our model, which represents the maximum number of trees during the boosted leaf-wise training of the algorithm. The *reg_alpha* and *reg_lambda* values corresponding to L1 and L2 hyperparameters, respectively. Both were set to 1e−2. Finally, the learning rate was set to 0.1 to boost the algorithm's learning process to ignore unwanted noise.

CB is another tree-based algorithm which supports multiclass classification and is fully compatible with sklearn's tools. CB is specially designed to deal with overfitting via two different detectors, namely *IncToDec* and *Iter*, that check the training parameters and if the pre-set thresholds of loss are exceeded the training process is stopped. For the needs of this work, through various repeated try and learn attempts the number of iterations was decreased to 70 to minimize the average time of the model's training. Also, as recommended, when the *max_iter* value is decreased the *learning_rate* should be increased, and for that reason it was set to 0.3. During the training of the algorithm, the CrossEntropy or negative log likelihood loss function was applied due to the multiclass nature of the LMD-2023 dataset to mitigate prediction errors. Moreover, *L2 (reg_lambda)* regularization parameter was set to 1e−7 to mitigate the cost of the loss function during the training of the model.

Lastly, Bagging is an ensemble meta-estimator that fits classifiers as base algorithms on random samples of the original dataset. Then, through averaging or voting, it assembles their distinct predictions into a final result. Its main advantage is that it reduces the variance of the model through the examination of multiple models. In this work, Bagging was applied initially with four different base estimators exclusively, namely DT, RF, ET and CB. Accordingly, the estimators parameter, which is related to the number of implemented algorithms or how many times a distinct estimator is going to be executed, was set to 12. To ensure the uniqueness of the sample in each run of the base estimators, the *random_state* value was set to 22. Moreover, the number of the samples to draw from the LMD-2023 dataset to train each base estimator was set to 80%, leaving the rest 20% of the data for testing each fit. Finally, the four aforementioned base estimators were implemented to be run consecutively within a final execution of the Bagging aggregated average model.

5.1.2 Results

Table 11 groups the shallow classification results per classifier, where the average score per metric was calculated over the 10 stratified cross evaluation folds. Specifically, the table contains the most relevant per classifier evaluation scores, namely, AUC, Precision, Recall, $F1$, Accuracy and Total Execution Time (T.E.Time) in days/h/min/s format. Recall that due to the imbalanced nature of the LMD dataset, the Accuracy metric is presented only for consistency and completeness purposes, therefore it is highlighted in italics. Instead, our focus is on the AUC and $F1$ metrics, where the highest and lowest average scores are presented in bolditalics and bold, respectively. The same colors are used for the fastest and slowest time, respectively. Figure 2 provides extra insight regarding the above-mentioned analysis and results, through the exported average versions of confusion matrices per ML model.

As observed from Table 11, the best mean results were achieved with the ET algorithm, achieving a mean AUC of 99.84%. Bagging (with DT algorithm as Base Estimator) and RF classification models succeeded the second and third-best average AUC-score, with 99.80% and 99.68% percentage each. As it concerns the rest of tree-based clas-

Table 10 Hyperparameter values per classification algorithm

Hyperparameters	Probabilistic	STH-based	Deter.	L-based	Tree-based					
	Log.Reg	SGDC	KNN	LSVC	LGBM	DT	RF	ET	CB	BNG
solver	*saga*	–	–	–	–	–	–	–	–	–
max_iter	1000	5e−3	–	30000	–	–	–	–	70	–
tol	1e−3	1e−8	–	–	–	–	–	–	–	–
C	–	–	–	1.5	–	–	–	–	–	–
loss	–	*m_huber*	–	–	–	–	–	–	*C.E.*	–
early_stopping	–	*True*	–	–	–	–	–	–	–	–
learning_rate	–	*Optimal*	–	–	0.1	–	–	–	0.3	–
algorithm	–	–	*auto*	–	–	–	–	–	–	–
leaf_size	–	–	50	–	–	–	–	–	–	–
n_neighbors	–	–	10	–	–	–	–	–	–	–
weights	–	–	*uni*	–	–	–	–	–	–	–
max_bin	–	–	–	–	20	–	–	–	–	–
max_depth	–	–	–	–	20	20	20	300	–	–
min_child_samples	–	–	–	–	30	–	–	–	–	–
min_data_in_bin	–	–	–	–	10	–	–	–	–	–
min_split_gain	–	–	–	–	0.1	–	–	–	–	–
multi_class	*multinomial*	–	–	–	–	–	–	–	–	–
n_estimators	–	–	–	–	100	–	300	1000	–	12
penalty	*elasticnet*	*elasticnet*	–	–	–	–	–	–	–	–
random_state	–	–	–	–	30	–	–	–	–	22
num_leaves	–	–	–	–	20	–	–	–	–	–
reg_alpha	–	–	–	–	1e−2	–	–	–	–	–
reg_lambda	–	–	–	–	1e−2	–	–	–	1e−7	–
n_jobs	–	–	–	–	8	−1	−1	−1	–	–
ccp_alpha	–	–	–	–	–	1e−3	1e−3	1e−2	–	–
max_leaf_nodes	–	–	–	–	–	100	100	650	–	–
max_samples	–	–	–	–	–	–	–	–	–	0.8
min_samples_leaf	–	–	–	–	–	8	8	8	–	–
min_samples_split	–	–	–	–	–	–	20	10	–	–
criterion	–	–	*mski*	–	–	*entropy*	*gini*	*gini*	–	–

A hyphen denotes that the current value either is inapplicable to the current algorithm, or if it is applicable, it implements the default value. STH-based, stochastic-based; L-Based, Linear-based SVC; Deter., Deterministic; C.E., C.Entropy; uni, uniform; mski, minkowski

sifiers, LightGBM and DT succeeded also an average AUC score above 99% (precisely 99.61% and 99.53% each), while CatBoost did not manage in all rounds of the 10-folds to overcome 98.09%. As shown in the same table, Bagging was executed with three more base estimators, namely RF, ET, and CB, giving a bagging tree-based average AUC score of 99.22%. On the other hand, Stochastic-based classification models, namely Logistic Regression, SGDC, KNN, and NB, did not manage to exceed 95% as it concerns the two former algorithms (93.51% and 93.27%, respectively) and 98.5% regarding the two latter (98.33% and 98.35% each). The worst performance was presented with LinearSVC algorithm, yielding an average AUC of 92.89%.

Regarding the $F1$ score, ET had, as it was expected from its presented in Sect. 5.1.1 core structure, the best rate with 99.41%, while SGDClassifier had the worst, i.e., 91.21%. Naive Bayes was the fastest algorithm, with $\approx$ 30 min of training time, while KNN revealed the maximum delay during the training of its model with $\approx$ 26 hr.

5.2 Deep learning

For DNN analysis, five different networks were created, namely MLP, CNN, LSTM, RNN and Autoencoders. MLP models, are feed-forward artificial NN that generate multiple interconnected dense layers for directing multiple input nodes through binary or multiclass graph-based classification into the corresponding output layers. Through the creation of perceptron models, researchers can handle complex classification experiments relying on the weighting of coefficients

Table 11 Results of shallow analysis

Model	AUC	Prec	Recall	$F1$	*Acc*	T.E.Time
Logistic regression	93.51	98.66	98.72	98.68	*98.72*	00:00:45:50
SGDClassifier	93.27	92.49	90.03	**91.21**	*98.47*	00:00:26:59
KNN	98.33	96.77	97.57	97.16	*99.47*	**01:01:02:00**
Naive Bayes	98.35	98.21	95.85	96.63	*95.85*	***00:00:30:29***
LinearSVC	**92.89**	95.16	89.45	92.01	*98.67*	00:01:20:13
LightGBM	99.61	99.38	99.36	99.37	*99.90*	00:00:32:19
DT	99.53	98.70	99.33	99.01	*99.82*	00:00:37:09
RF	99.68	98.61	99.56	99.08	*99.83*	00:01:45:05
ET	***99.84***	99.05	99.79	***99.41***	*99.89*	00:07:30:12
CatBoost	98.09	98.32	97.17	97.73	*99.59*	00:00:25:48
Bagging (Base Estimator: DT)	99.80	99.04	99.73	99.38	*99.88*	00:01:23:33
Bagging (Base Estimator: RF)	99.37	98.50	99.10	98.78	*99.77*	00:00:30:05
Bagging (Base Estimator: ET)	99.72	98.88	99.61	99.24	*99.86*	00:01:56:30
Bagging (Base Estimator: CB)	97.99	98.28	97.04	97.64	*99.57*	00:05:43:30
Bagging (Tree-Based Avg.)	99.22	98.67	98.87	98.76	*99.24*	00:03:38:25

T.E.time, Total execution time

to evaluate numerous inputs. MLP models are widespread due to their versatile nature to handle non-linear and highly imbalanced datasets and be trained in parallel real-time mode keeping high $F1$ score percentages, even via small samples.

On the other hand, CNN is a DNN algorithmic model that takes advantage of the implementation of various convolutional and pooling layers. Convolutional layers are responsible for the apportionment of large scaled imbalanced sets of data into various areas of nodes, whereas the pooling creates a "pool" with the highest values in each area. Despite their best applicability in unsupervised learning experiments, they are equally effective algorithms for supervised learning as well, taking into account the existence of large scaled datasets with millions of records.

As it concerns LSTM networks, they are an elevated form of the so-called Recurrent Neural Networks (RNN). They are designed to train their models and learn from long-term dependent features through the gated recurrent units (GRUs) method. GRUs allows the manipulation of two gates, one "forget gate" that intentionally omits information from previous timesteps and the "update gate", which is responsible to calibrate the amount of information that will be fed into the next timestep. Among the principal advantages of LSTM is its efficiency to classify long sequences that emerge from the existence of numerous features and extended sets of data.

Simple RNN models were initially associated with the predictive classification of data samples that present sequential behavior, such as voice, language, or image datasets. Despite their initial dedicated implementations, they have found applicability in the IDS domain with highly promising results, as presented in [29–31]. Finally, autoencoders is another popular type of feed-forward artificial NN. Their functionality hinges on the design that the input is compressed and downgraded into a low-dimensional coded format and then is reconstructed from this representation. Similar to the majority of the aforementioned DNN algorithms, autoencoders have been initially applied over picture and human figures recognition. However, as evidenced in [32–37], such models can provide high prediction rates in IDS concepts too.

The above-mentioned models were built on the same desktop machine as described in Sect. 5.1. Nevertheless, to speed up the training process, all the experiments were run on the base-machine's GPU and RAM, via NVIDIA's cuDNN 8.1.0.77 library for GPU-acceleration and parallel programming. Despite the use of the CUDA GPU implementation library, the relevant to the experiments scripting employed Python 3.10.9 via Sklearn 1.2.1, Pandas 1.5.2, Numpy, 1.24.2, Seaborn 0.12.2, matplotlib 3.4.3, jupyter_server 8.0.3, Keras 2.10.0 and Tensorflow 2.10.1 libraries and packages.

5.2.1 Configuration of hyperparameters

Table 12 delineates the hyperparameters used for the creation of each of the DNN models. For allowing the maximum level of coefficients balancing during the training phase of CNN, LSTM, and Autoencoders, the robustness of stochastic gradient descent was calibrated via the selection of the mini-batch SGD optimizer and through a *momentum* of 0.9 and a *learning rate* of 2e−2. Regarding the MLP and RNN models, another variant of the classic SGDC algorithm was implemented, namely *Adam* optimizer. Although the original SGDC optimizer maintains a constant *alpha* learning rate for all weight updates during training, *Adam* maintains a per-parameter learning rate, which makes it very efficient when the datasets have very large diversions in gradients. Considering the fact that a balance between the number of the executed per NN *epochs* and the number of samples that will be passed through to the network at one time, namely the *batch size*, is crucial for achieving the best generalization during the training phase of the examined samples, an extended trial and error approach was adopted. This approach concluded in the use of a *batch size* of 50 and 32 for MLP (40 epochs), CNN (20 epochs) and LSTM (12 epochs) models, respectively. Lastly, both RNN and Autoencoders models were initialized via the use of a *batch size* of 50.

The popular *relu* activator was exploited in MLP, CNN, and Autoencoders, whereas the *tahn* activator was implemented as compatible with the LSTM and RNN networks. Both *relu* and *tahn* activators were used only for the input and the various hidden layers of each network. For the output layer, which is responsible for the classification of each sample, the *Softmax* output activator was implemented as proposed in the majority of the studied bibliography. In order to make the training of DNN networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling, an extra *batch normalization layer* was implemented as hidden in each algorithm. As an extra regularization effect, the *Dropout* technique was used to prevent overfitting. The *Dropout* parameter was set to 0.8 and 9.0 as presented in Table 12.

As it concerns the input layer of each NN presented in Table 12, it was the same for all models, namely 90 columns. It should be pinpointed that the extended number of columns was the cause of the OHE preprocessing of the LMD-2023 dataset, as explained in Sect. 4.2. The output layer was set to outline the three classes of the dataset's labelling, as given in Table 6. The provided by Keras embedding layer was also implemented right after the input layer of the CNN and LSTM models.

The CNN model was used with six hidden *Conv1D* layers, and one *Flatten* after the last *Conv1D*. Padding was set to *"same"* in all six *Conv1D* layers. On the other hand, LSTM was implemented as a combination of five hidden layers and one *GlobalAveragePooling1D* after the last hidden. Above that, a *BatchNormalisation* layer was implemented after each of the five hidden layers.

5.2.2 Results

Table 13 recapitulates the DNN classification results of the LMD-2023 dataset. As with the swallow classification results in Table 11, the numbers represent the average score per metric, as that were calculated over the 10 stratified cross validation folds, taking into account the total of the executed epochs per algorithm. Again, due to the highly imbalanced nature of the LMD dataset, the Accuracy score is highlighted in italics and is presented only for reasons of consistency and completeness. The results concentrate on the AUC and $F1$ scores, from which the best and worst average scores are presented in bolditalics and bold, respectively. Additionally, Sect. 6 contains the results of each confusion matrix, as depicted in Figs. 2 and 3, respectively.

With reference to Table 13, the best average AUC score was achieved by the LSTN algorithm, with an average value of 95.82%, which is also justified by the best average $F1$ score, regarding the same model and with an average value of 95.55%. On the downside, this DNN model was the slowest in terms of training time, i.e., $\approx$ 16 hr. The CNN model presented the worst performance in terms of AUC score, with 95.12%, whereas MLP had the worst $F1$ score with a value of 93.57%. Lastly, the MLP model was the fastest, during the training time, requiring $\approx$ 11 h. Regarding the number of optimal epochs per cross-validated fold were very high, as its rate varied from an average of 52 for MLP model, to 32 and 30 for CNN and LSTM networks, respectively. Altogether, with reference to Table 13, it is interesting to see that the two new DNN models present a similar score to CNN (94.87% and 94.55% vs. 94.43%, respectively) and they are quite close (−0.68% and −1%) to the best DNN performer, namely LSTM. However, both these algorithms presented a significant latency in terms of their total training and prediction time, with an average of $\approx$ 15 h of execution.

6 Discussion

Following the first stage observations on the results given in Tables 11 and 13, the current section encloses a deeper analysis of the overall findings, also vis-à-vis the related work of Sect. 2.

6.1 Shallow classification

As presented in Table 11, ET achieved the best AUC and $F1$ scores; 99.84% and 99.41%, respectively. On the other hand, LinearSVC presented the worst AUC score, i.e., 92.89%, while SGDClassifier had the worst $F1$ score with 91.21%. Overall, the disparities observed for the metrics of the same type for both type of analysis are highly diverged regarding the best and the worst shallow classification average. For instance, 6% to 7% difference between the best and the worst average AUC: 99.84% (ET) and 92.89% (LinearSVC), whereas 0.5% to 0.8% for DNN analysis best and worst collected rates, namely 95.82% (LSTM) and 95.12% (CNN). The same divergence is also observed regarding the $F1$ score: between 99.41% (ET) and 91.21% for shallow, and 95.55% and 93.57% for DNN. Tree-based algorithms performed better in shallow classification analysis, compared to

Table 12 Parameter values per DNN algorithm

Hyperparameters	MLP	CNN	LSTM	RNN	Autoencoders
Activator	relu	relu	tanh	tanh	relu
Input_dim	90	90	90	90	90
Output activator	softmax	softmax	softmax	softmax	softmax
Initializer	he_normal	–	–	–	–
Optimizer	adam	sgd	sgd	adam	sgd
Momentum	0.8	0.9	0.9	0.9	0.9
Dropout	0.15	0.2	0.3	0.15	0.2
Learning rate	1e−2	2e−2	2e−2	1e−2	1e−2
Loss	CCE	CCE	CCE	SCC	SCC
Reg.l2	1e−2	1e−2	2e−2	1e−2	1e−2
Batch Norm	Yes	Yes	Yes	Yes	Yes
Embedding layer	No	Yes	Yes	No	No
Flatten layer	No	Yes	Yes	No	No
Standardization	No	Yes	Yes	No	No
Hidden layers	4	6	5	4	5
Nodes (Per layer)	30/20/10/5	50/30/20/15/10/5	50/30/20/10/5	30/20/10/5	50/30/20/10/5
Epochs	40	20	12	40	35
Batch size	50	32	30	50	50

The Dropout parameter's value of 0.15 was applied in the final hidden layer for the total of the depicted DNN algorithms. The hidden layers values are calculated without including the input and output layer. CCE, Categorical CrossEntropy loss function; SCC, SparseCategoricalCrossentropy loss function; Input_dim, number of the features used as input. A hyphen defines a non-applicable option for this DNN model

Table 13 Results of DNN classifiers analysis

Model	AUC	Prec.	Recall	$F1$	*Acc*	Epochs	T.E.Time
MLP	95.57	95.05	92.22	**93.57**	*98.87*	40	***00:10:52:46***
CNN	**95.12**	95.44	92.68	94.43	*98.66*	20	00:13:28:20
LSTM	***95.82***	95.11	94.36	***95.55***	*98.93*	12	**00:15:44:18**
RNN	95.26	95.64	93.12	94.87	*98.83*	40	00:14:56:13
Autoencoders	95.10	95.17	92.14	94.55	*98.24*	35	00:15:32:28

T.E.Time, Total execution time

the results of the other four categories of Table 10 with L-based algorithm to be the worst performer. Precisely, all the six Tree-based algorithms in total yielded an at least 4% better $F1$ score vis-à-vis the other four categories of classifiers.

With reference to the confusion matrices of Fig. 2, LightGBM presented superior results. That is, the algorithm successfully identified 402,533 logs (or else 90.57% TP rate) related to normal traffic, plus another $\approx 41,664$ logs, revealing 9.37% of TN results. Overall, it achieved a score of 99.94% regarding the identification of Normal network traffic, as the sum of TP and TN rates. On the downside, LightGBM misidentified 140 logs as normal and another 146 as malicious, revealing a minor tendency of 0.25% and 0.23% on FP and FN events, respectively regarding Normal samples labelled as "0". The EoRS (labeled as "1") class suggested Bagging (Tree-Based Avg.) as the best performer; $\approx$ 31 (0.006%) samples were misclassified. As it concerns the EoHT (labeled as "2"), the best rate regarding this label exclusively was achieved with the LightGBM algorithm; $\approx$ 418 (0.09%) events were misidentified. In general, 11 out of the 15 executed shallow classifiers of Table 11 managed to predict a minimum of $\approx$ 98%, of both TP and TN values, with respect to the EoRS and EoHT classes, respectively. On the flip side, SGDClassifier, LinearSVC, and Naive Bayes, misclassified 6,926 (or 18.55%), 6,014 (or 16.22%) and 18,829 (or 45.52%) samples from the EoRS and EoHT classes, respectively. This behavior leaves promising perspectives for future work to consider the capabilities and the general behavior of the three aforementioned ML models. More specifically, via the combination of artificially created custom features the foregoing disparities are aggregated between the best and worst cross validation metrics case, around a generally accepted mean value.

Table 14 Comparison with related work (all metrics are in %)

Model	Features	Cls	AUC	Prec	Recall	$F1$	*Acc*	Epochs	Bal.	k-fold	T.E. Time
Lateral movement previous works											
MV (RF, LB, LoR) [7]	4	2	–	–	–	0.66	*99.62*	N/A	×	✓	–
GRU DNN [12]	8	2	–	93.23	–	–	*96.68*	60	×	✓	-
Ensemble ML [13]	8	2	–	88.70	–	–	–	N/A	×	✓	-
SS DL [17]	8	2	–	91.3	–	–	*99.9*	N/A	✓	×	-
UML with JD [18]	15	2	–	6	–	–	-	N/A	×	×	-
K-Means UML [19]	27	2	81	–	–	–	–	N/A	✓	×	-
RF [11]	29/17*	2	-	83.73	81.23	0.82	–	N/A	×	✓	00:00:02:06
LaBi [10]	32	2	-	99.87	99.47	0.97	*99.99*	N/A	×	✓	00:00:11:28
RF [9]	35	2	–	80.31	80.29	0.8	–	N/A	✓	✓	00:00:03:11
This work. In each case, the best performers based on F1 score for LMD dataset											
ET	15	3	***99.84***	99.05	99.79	***99.41***	*99.89*	N/A	✓	×	00:07:30:12
LSTM	15	3	***95.82***	95.11	94.36	***95.55***	*98.93*	30	✓	×	**00:15:44:18**

An “*” denotes the parameter through which the best result was achieved. The best/worse performers shown in bolditalics/bold for this work are with reference to the $F1$ metric. Dash, not provided; N/A, not applicable; Bal., balanced test set; Cls, number of classes considered

Table 15 Presentation of the prediction rates of DNN algorithms per class

Model/Class	Prec			Recall			$F1$			Acc.		
Class	0	1	**2**	0	1	**2**	0	1	**2**	0	1	*2*
MLP	99.20	99.99	**85.95**	99.58	99.04	**78.02**	99.39	99.51	**81.79**	98.72	98.89	*98.97*
CNN	99.26	99.99	**87.42**	99.62	99.05	**79.86**	99.44	99.52	**83.04**	99.01	98.25	*98.99*
LSTM	99.29	99.99	**83.88**	99.43	99.06	**82.83**	99.36	99.52	**83.76**	98.43	98.96	*98.80*

The Accuracy scores regarding all three classes, but more precisely the EoHT class labeled as “2”, is highlighted with italics and is ignored due to the imbalanced nature of the dataset

6.2 DNN

As regards DNN analysis, with reference to Table 13, the best performer was LSTM. As already mentioned in Sect. 5.2.2, this model achieved an average score of 95.82% and 95.55% regarding AUC and $F1$ score, respectively. Although it was the slowest, it required the minimum number of epochs before the early stopping parameter terminates the training process due to lack of evolution. LSTM misidentified 4,312 (or 0.97%) samples of the Normal class (labeled as “0”), while the MLP and CNN missed $\approx$ 5,000 (1.12%) regarding the samples of the same class. The MLP presented the worst prediction percentage for the Normal class, i.e., more than 4,900 or 1.1% misplaced samples. Regarding the EoHT class, LSTM presented almost the same results as with the Normal class, as 4,305 (0.96%) of the samples were misidentified. MLP and CNN misplaced an average of $\approx$ 4,800 (1.08%) samples while, RNN and Autoencoders misplaced an average of $\approx$ 4,500 or 0.97% samples. Specifically, with reference to Fig. 3, all five algorithms thrived in the EoRS class, misplacing only about 236 or 0.04% of the samples.

6.3 Comparison with related work

This section complements Sect. 2 by providing a deeper comparison with the related work. Table 14 gathers the common characteristics, including the methodology, number of classes, features and metrics of major past works, considering supervised and unsupervised ML analysis of LM events. Moreover, the table includes only the best performers per ML analysis, as given in Sects. 5.1 and 5.2. The contributions [20–23] of Sect. 2.3 referring to graph-based ML, along with the work in [14] respecting to unsupervised ML techniques, were omitted for reasons of inconsistency regarding the presentation of the conducted experiments and the collected results from them. Precisely, the former works approach the subject of LM techniques thought the graph-based perspective, and although they demonstrate promising results they are incompatible with our study. Regarding the work in [14], it was purposefully omitted because it focuses on insider threat ML analysis, rather than log-based intrusion detection.

Taking into account that almost half of the depicted in Table 14 works do not include a completed column of metrics, the rows of the table are sorted based on the number of features, presented in the second rightmost column. Further,

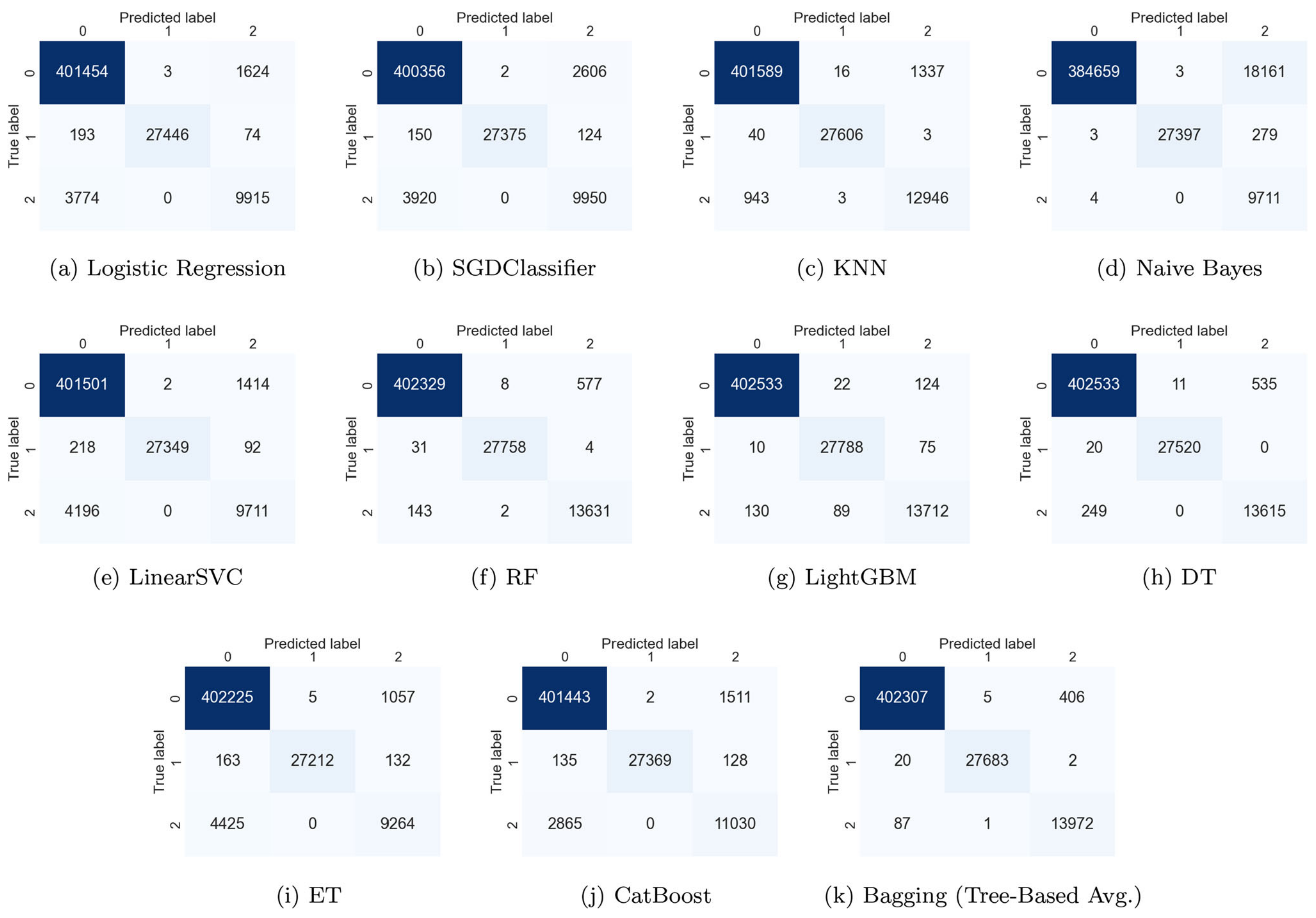
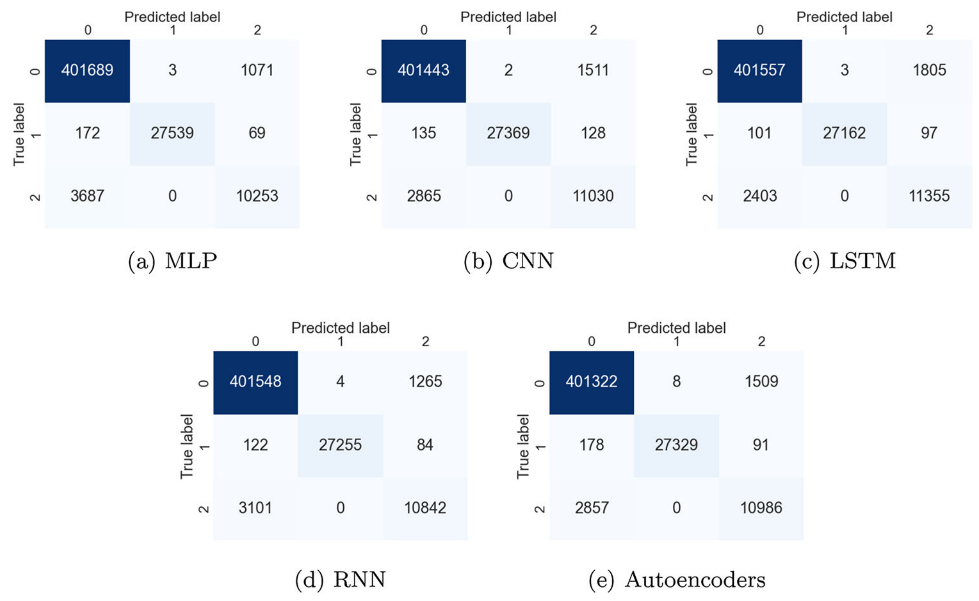


Fig. 2 Confusion matrices for the shallow classification experiments of Sect. 5.1 on the LMD-2023 dataset. Average results over all folds for shallow model analysis

Fig. 3 Confusion matrices for the DNN classification experiments of Sect. 5.2 on the LMD-2023 dataset. Average results over all folds for shallow model analysis



given that all the included works consider binary classification models, while the current takes into account three classes as they have been defined in Sect. 3.4 it might seem reasonable for the experiments of Sect. 5 to be repeated under the concept of two classes, namely Normal and Malicious. However, as it was proven experimentally both during the labelling of the dataset and the designing of shallow and DNN experiments, in Sects. 3.4 and 5, respectively, the multiclass classification analysis takes more time to run than a binary classification task. For that reason, the relationship between the number of classes and the runtime is considered roughly linear and the ML problem with more classes added is NP-hard to be solved, requiring most of the CPU's computing power resources.

As observed from Table 14, most contributions omit to report key evaluation metrics as part of the documentation of their methodology; this is evident especially for the AUC, Recall, and $F1$ scores. Moreover, six out of nine works included k-fold cross validation (all with 10 folds) as a precaution step for avoiding overfitting. Additionally, the majority of the contributions employ an imbalanced dataset, whereas each one of the works [9, 17, 19] incorporate balanced traffic. Nevertheless, it is generally accepted that the imbalanced nature of real-life log-based traffic is the basic principle that governs real-world scenarios. Put simply, it is almost unlikely to be numerical equal malicious with normal logs, and in many cases, attack logs may surpass Normal ones. Think for instance the exploitation of remote services, where numerous attempts are executed towards the acquisition of remote access to the targeted host.

Furthermore, all works except [12] do not make any reference to the features that were employed as inputs to their examined ML model, the following st was left from previous revision and included in the submitted manuscript even though the majority of the studies rely on features that were artificially extracted as custom and not generic ones, not to mention that none of them or the rest of the related works presented in Table 1 deal with Sysmon oriented logs. This lack of information, regarding one of the keystone aspects of ML analysis, deprive knowledge from future researchers in the subject of LM analysis to draw important conclusion of how overfitting issues could be prevented through feature's manipulation and preprocessing.

It should be pinpointed that none of the works in Table 14 includes regularization of ML mode's hyperparameter when those are implemented in their methodology. Precisely, when it comes to hyperparameter tuning it refers to the identification of the optimal setting of each algorithm and is completed in advance the initiation of the ML process and the construction of the model. It is a trial and correct procedure that has been proved through practice as the first step towards calibrating the speed and quality of the learning process and the overall performance of the model being trained. Above that, hyperparameter tuning is equally critical when it comes to the construction of DNN layered models, when the number of layers and their included parameter settings affect exponentially the learning process and the overall prediction of the network. Equally important, finding the optimal balance of hyperparameter can eliminate overfitting, due to too low or very high learning that cause overfitting via under-sampling or collisions, respectively.

From Table 14, it is derived that the authors of the related works heavily rely on the Precision metric for assessing their ML models. Besides that, only four papers introduce Accuracy as their assessing metric, among which only one considers a balanced dataset. Nevertheless, although the avoidance of Accuracy towards the evaluation of imbalanced datasets is sound in favor of Precision, this metric should also be weighted in conjunction with Recall results and under the metrics of AUC and $F1$. For instance, although the contributions in [7, 10] report a high accuracy rate of 99.62% and 99.99%, respectively, they demonstrate a low averaged prediction rate of 0.66 and 0.97, respectively, regarding the $F1$ score. Based on our results, ET yielded a prediction rate of 99.41% for $F1$, while the LSTM model had a prediction score of 95.55%. Both models were selected as the best in terms of $F1$ score and AUC combination, as depicted for shallow and DNN analysis in Sect. 5.1.2 and Table 13, respectively. Once again, it is important to clarify that due to the imbalanced nature of the LMD-2023 dataset, the $F1$ and AUC metrics should be the primary focus in any related to this subject ML analysis.

Overall, compared to the limited so far work in the field of ML analysis on LM events, and taking into account that, to the best of our knowledge, for the first time a contribution harnesses Sysmon logs through supervised shallow and DNN elevated classification techniques, the results given by the current study are superior and more than promising. Recall from Sect. 3.1 that Sysmon logs supersede that of the legacy MS Windows Event Viewer, being much richer and appropriate for conducting dataset creation and feature selection towards the experimentation over ML techniques.

6.4 Takeaways and future directions

Based on both theory and extracted empirical observations from the conducted experiments, this work provides a unique, to our knowledge so far, methodology (alongside an open-source, publicly available tool called ETCExp) that exploits Sysmon logs towards the detection of LM. As already pointed out in Sect. 3.2, the creation of ETCExp was motivated by the lack of the literature to offer an open-source tool that covers the needs of EVTX-to-CSV transformation and extraction. ETCExp not only extracts the manipulated EVTX files into CSV equivalent, but on demand conducts the dataset's labelling procedure, as detailed in Sect. 3.4. This is done based on the re-configurable policy against LM events given in [24]. According to the related work in Sect. 2, no study except [12] incorporated logs extracted from Sysmon captured traffic under the concept of supervised classification. Nevertheless, opposite to the study at hand, in [12] the

Sysmon logs analysis was conducted under a binary classification scheme and not under a multiclass labelled dataset.

Moreover, the current work provides a stable methodology for feature selection and data-preprocessing targeting Sysmon's log-based datasets. Although this study considers a rather small but impactful set of 15 features, depending on the complexity of the LM incident under investigation, the number of the features may be altered to a larger or smaller set. A promising path left as a goal for future work will be the implementation of custom features, such as a counter for the number of specified to targeted LM techniques Sysmon headers.

Another promising future path would be the implementation of unsupervised ML techniques in combination with regression methods towards the successful identification of the unlabeled LMD (or any other similar) dataset version. We argue that the combination of the three pillars of the unsupervised ML, namely Clustering, Association and Dimensionality reduction, will reveal useful relationship patterns on the collected LM-related samples. Upon that, each sample's dimensional interdependence may be reduced towards the extraction of artificial custom features. Moreover, the examination of Regression ML models via Supervised-ML (SML) or Unsupervised-ML (UML) may assist in the implementation of new numerical features, similar to the six features preprocessed with MinMax scaling in Table 8. This may lead to the creation of other, more generalized ML models.

As it concerns LMD-2023 DNN analysis, it was observed that the number of the collected samples related to the EoHT class were not quite enough to train the DNN models to produce superior predicting ratings. Regarding the EoHT class, as it is presented in Table 15, although the MLP model it was trained for 50 epochs it did not manage to exceed the average score of $\approx 82\%$, as it concerns the three major metrics, namely Precision, Recall and $F1$. The analysis revealed the same approximate results on behalf of CNN and LSTM networks, a fact that makes clear the incapability of the model to generalize well on such a rather small subset of samples ($\approx 30.5K$ samples in a total of $\approx 1.7M$). Naturally, future work may expand the LMD dataset to include more samples for certain attack classes in an effort to further improve the prediction score of DNN models.

7 Conclusions

The present study aspires to set the groundwork for a comprehensive IDS approach regarding the detection of LM through ML techniques; the focus is on the MS Windows platform and the Sysmon system service. In comparison to the relevant literature, which is rather scarce, we offer a solid, all-encompassing process for selecting the most appropriate classification features and data preprocessing methods towards the elimination of overfitting and the implementation of accurate and well-generalized ML models. It is demonstrated that datasets created from the extraction of Sysmon logs into the CSV format can be particularly effective, even in multiclass classification, if just trained with a bare minimum of high importance features. Notably, the detection score in terms of the $F1$ metric is above 99% for almost half of the shallow estimators we tested and above 94.4% for all but one of the utilized DNN models. Just as importantly, we pinpoint shortcomings or misconceptions observed in the related work, suggesting the right direction.

Following the discussion in Sect. 6.4, future work can assess the ability of selecting the same features under the concept of unsupervised ML techniques. We argue that the implementation of the three state-of-the-art directions of UML, namely Clustering, Association and Dimensionality reduction, will reveal useful relationship patterns on the collected LM-related samples. Besides, the combination with regression methods exploiting unlabeled datasets, will also offer greater insight into the LM ecosystem. Finally, a straightforward direction from future work is the enrichment of LMD family of datasets with more LM techniques that will trigger the generation of a richer set of EventIDs.

Funding Open access funding provided by HEAL-Link Greece. This study received no funding.

Data availability All data and code generated or used to support the findings of this study are included within the article.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest regarding the publication of this study.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Makrakis, G.M., et al.: Industrial and critical infrastructure security: technical analysis of real-life security incidents.

IEEE Access **9**, 165295–165325 (2021). https://doi.org/10.1109/ACCESS.2021.3133348
2. González-Manzano, L., et al.: A technical characterization of APTs by leveraging public resources. Int. J. Inf. Secur. (2023). https://doi.org/10.1007/s10207-023-00706-x
3. MITRE: Lateral movement—the adversary is trying to move through your environment (2019)
4. Sarah Hawley - Ben Read - Cristiana Brafman_Kittner - Nalani Fraser - Andrew Thompson - Yuri Rozhansky - Sanaz Yashar. APT39—An Iranian Cyber Espionage Group Focused on Personal Information (2021)
5. Corfield, G.: SolarWinds hack was done by Kremlin's APT29 crew, say UK and US (2021)
6. Gillis, T., et al.: Lateral movement in the real world—a quantitative analysis (2022). https://blogs.vmware.com/security/2022/06/lateral-movement-in-the-real-worlda-quantitative-analysis.html. Visited on 2022
7. Kaiafas, G., et al.: Detecting malicious authentication events trustfully. In: NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–6 (2018). https://doi.org/10.1109/NOMS.2018.8406295
8. Kent, A.D.: Cybersecurity data sources for dynamic network research. In: Dynamic Networks in Cybersecurity. Imperial College Press (2015)
9. Bian, H. et al.: Host in danger? Detecting network intrusions from authentication logs. In: 2019 15th International Conference on Network and Service Management (CNSM), pp. 1–9 (2019). https://doi.org/10.23919/CNSM46954.2019.9012700
10. Bai, T., et al.: A machine learning approach for RDP-based lateral movement detection. In: 2019 IEEE 44th Conference on Local Computer Networks (LCN), pp. 242–245 (2019). https://doi.org/10.1109/LCN44214.2019.8990853
11. Bian, H., et al.: Uncovering lateral movement using authentication logs. IEEE Trans. Netw. Serv. Manag. **18**(1), 1049–1063 (2021). https://doi.org/10.1109/TNSM.2021.3054356
12. Chen, C.-M., Syu, G.-H., Cai, Z.-X.: Analyzing system log based on machine learning model. Int. J. Netw. Secur. **22**(6), 925–933 (2020)
13. Bohara, A., et al.: An unsupervised multi-detector approach for identifying malicious lateral movement. In: 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), pp. 224–233 (2017). https://doi.org/10.1109/SRDS.2017.31
14. Le, D.C., Zincir-Heywood, N.: Anomaly detection for insider threats using unsupervised ensembles. IEEE Trans. Netw. Serv. Manag. **18**(2), 1152–1164 (2021). https://doi.org/10.1109/TNSM.2021.3071928
15. Center, C., Trzeciak, R.: The CERT insider threat database. In: Carnegie Mellon University's Software Engineering Institute Blog (2011)
16. Harilal, A., et al.: TWOS: a dataset of malicious insider threat behavior based on a Gamified competition. In: Proceedings of the 2017 International Workshop on Managing Insider Security Threats. MIST '17. Association for Computing Machinery, Dallas, Texas, USA, pp. 45–56 (2017). ISBN: 9781450351775. https://doi.org/10.1145/3139923.3139929
17. Chen, M., et al.: A novel approach for identifying lateral movement attacks based on network embedding. In: 2018 IEEE international conference on parallel & distributed processing with applications, ubiquitous computing & communications, big data & cloud computing, social computing & networking, sustainable computing & communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), pp. 708–715 (2018). https://doi.org/10.1109/BDCloud.2018.00107
18. Bhasin, H.P.S., et al.: Data center application security: lateral movement detection of malware using behavioral models. SMU Data Sci. Rev. **1**(2), 10 (2018)
19. Powell, B.A.: Role-based lateral movement detection with unsupervised learning. Intell. Syst. Appl. **16**, 200106 (2022)
20. Purvine, E., Johnson, J.R., Lo, C.: A graph-based impact metric for mitigating lateral movement cyber attacks. In: Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense. SafeConfig '16. Association for Computing Machinery, Vienna, Austria, pp. 45–52 (2016). ISBN: 9781450345668. https://doi.org/10.1145/2994475.2994476
21. Liu, Q., et al.: Latte: large-scale lateral movement detection. In: MILCOM 2018—2018 IEEE Military Communications Conference (MILCOM), pp. 1–6 (2018). https://doi.org/10.1109/MILCOM.2018.8599748
22. Ho, G., et al.: Hopper: modeling and detecting lateral movement. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, pp. 3093–3110 (2021). ISBN: 978- 1-939133-24-3
23. Fang, Y., et al.: LMTracker: lateral movement path detection based on heterogeneous graph embedding. Neurocomputing **474**, 37–47 (2022). https://doi.org/10.1016/j.neucom.2021.12.026. (**ISSN: 0925-2312**)
24. Smiliotopoulos, C., Barmpatsalou, K.: Revisiting the detection of lateral movement through Sysmon. Appl. Sci. (2022). https://doi.org/10.3390/app12157746. (**ISSN: 2076-3417**)
25. Smiliotopoulos, C., Barbatsalou, K., Kambourakis, G.: Python_Evtx_Analyzer (PeX - v1) (2022). https://github.com/ChristosSmiliotopoulos/Python_Evtx_Analyzer.git. Visited on 2022
26. Russinovich, M., Garnier, T.: Sysmon v13. 22. In: Retrieved 28 June 2021 (2021)
27. Smiliotopoulos, C., Kambourakis, G.: evtx_To_CSV_Export Tool (ETCExp) (2023). https://github.com/ChristosSmiliotopoulos/evtx_To_CSV_ExportTool. Visited on 2023
28. Smiliotopoulos, C., Kambourakis, G.: "LMD" Sysmon Dataset Collections (2023). https://github.com/ChristosSmiliotopoulos/Lateral-Movement-Dataset--LMD_Collections. Visited on 2023
29. Kasongo, S.M.: A deep learning technique for intrusion detection system using a recurrent neural networks based framework. Comput. Commun. **199**, 113–125 (2023). https://doi.org/10.1016/j.comcom.2022.12.010. (**ISSN: 0140-3664**)
30. Laghrissi, F., et al.: Intrusion detection systems using long short-term memory (LSTM). J. Big Data **8**(1), 65 (2021). https://doi.org/10.1186/s40537-021-00448-4
31. Tang, T.A., et al.: Deep recurrent neural network for intrusion detection in SDN-based networks. In: 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), pp. 202–206 (2018). https://doi.org/10.1109/NETSOFT.2018.8460090
32. Song, Y., Hyun, S., Cheong, Y.-G.: Analysis of autoencoders for network intrusion detection. Sensors (2021). https://doi.org/10.3390/s21134294. (**ISSN: 1424-8220**)
33. Singh, A., Jang-Jaccard, J.: Autoencoder-based unsupervised intrusion detection using multi-scale convolutional recurrent networks. In: CoRR abs/2204.03779 (2022). https://doi.org/10.48550/arXiv.2204.03779. arXiv: 2204.03779
34. Kamalov, F., et al.: Autoencoder-based intrusion detection system. In: 2021 International Conference on Engineering and Emerging Technologies (ICEET), pp. 1–5 (2021). https://doi.org/10.1109/ICEET53442.2021.9659562
35. Narayana Rao, K., Venkata Rao, K., Prasad Reddy, P.V.G.D.: A hybrid intrusion detection system based on sparse autoencoder and deep neural network. Comput. Commun. **180**, 77–88 (2021). https://doi.org/10.1016/j.comcom.2021.08.026. (**ISSN: 0140-3664**)
36. Chatzoglou, E., et al.: Pick quality over quantity: expert feature selection and data preprocessing for 802.11 intrusion detection systems. IEEE Access **10**, 64761–64784 (2022). https://doi.org/10.1109/ACCESS.2022.3183597

37. Chatzoglou, E., et al.: Best of BothWorlds: detecting application layer attacks through 802.11 and non-802.11 features. Sensors (2022). https://doi.org/10.3390/s22155633