electronics

MDPI

Article

Swarm Optimization and Machine Learning Applied to PE Malware Detection towards Cyber Threat Intelligence

Santosh Jhansi Kattamuri [1,2], Ravi Kiran Varma Penmatsa [2,*], Sujata Chakravarty [1] and Venkata Sai Pavan Madabathula [3]

[1] Department of Computer Science & Engineering, Centurion University of Technology & Management, Bhu-baneswar 761211, Odisha, India
[2] Department of Computer Science and Engineering, Maharaj Vijayaram Gajapathi Raj College of Engineering, Vizianagaram 535005, Andhra Pradesh, India
[3] Cisco Systems, Bengaluru 560103, Karnataka, India
* Correspondence: ravikiranvarmap@gmail.com

Abstract: Cyber threat intelligence includes analysis of applications and their metadata for potential threats. Static malware detection of Windows executable files can be done through the analysis of Portable Executable (PE) application file headers. Benchmark datasets are available with PE file attributes; however, there is scope for updating the data and also to research novel attribute reduction and performance improvement algorithms. The existing benchmark dataset contains non-PE header attributes, and few ignored attributes. In this work, a critical analysis was conducted to develop a new dataset called SOMLAP (Swarm Optimization and Machine Learning Applied to PE Malware Detection) with a value addition to the existing benchmark dataset. The SOMLAP data contains 51,409 samples that include both benign and malware files, with a total of 108 pure PE file header attributes. Further research was carried out to improve the performance of the Malware Detection System (MDS) by feature minimization using swarm optimization tools, viz., Ant Colony Optimization (ACO), Cuckoo Search Optimization (CSO), and Grey Wolf Optimization (GWO) wrapped with machine learning tools. The dataset was evaluated, and an accuracy of 99.37% with an optimized set of 12 features (ACO) proves the efficiency of the dataset, its attributes, and the algorithms used.

Keywords: malware detection; cyber threat intelligence; PE file; machine learning; Ant Colony Optimization (ACO); Cuckoo Search Optimization (CSO); Grey Wolf Optimization (GWO)

Citation: Kattamuri, S.J.; Penmatsa, R.K.V.; Chakravarty, S.; Madabathula, V.S.P. Swarm Optimization and Machine Learning Applied to PE Malware Detection towards Cyber Threat Intelligence. *Electronics* **2023**, *12*, 342. https://doi.org/10.3390/electronics12020342

Academic Editors: Ivan Cvitić, Dragan Peraković, Anca Delia Jurcut and Goran Marković

Received: 1 December 2022
Revised: 21 December 2022
Accepted: 28 December 2022
Published: 9 January 2023

1. Introduction

Malware is software which is intentionally created by adversaries to cause potential damage to an asset. Malware attacks have risen exponentially on par with technological advancements, including well-proliferated internet access and the Internet of Things. As malware dynamics are changing day by day, we need something sophisticated which can detect zero-day malware. Malware, as one of the largest threat vectors to security officials, poses a continuous challenge. Cyber threat intelligence (CTI) strategies involve gathering several data attributes, building profiles, using intelligent algorithms, and developing optimized threat detection and mitigation techniques. Malware sensing can be done either by static attribute analysis or by dynamically studying the source parameters. The machine learning (ML) approach to threat detection is highly reliable and efficient and supports both static as well as dynamic methods. One of the expected qualities of an efficient CTI is detection of zero-day attacks, and this can be achieved with ML-based detection. Windows is the most-used OS in the computer world; as of now, globally, 35.29% users use a Windows operating system [1]. Windows executable files are a possible threat vector that needs a regular threat screening procedure. Windows executable programs have a standard encapsulation format called the Portable Executable (PE) file standard. It is mandatory for

an executable windows file to be appended with a PE header [2]. PE file structure was initially explained by Pietrek [3]. One of the early path-setting works to detect malware by studying the PE file header attributes is by Schultz et al. [4], a static malware-sensing category. They also established the effectiveness of ML for malware identification using PE file header attributes. Some other early works that contributed in this direction are [5–7].

Overview of PE File Header Attributes

A Portable Executable (PE) file is the binary format for DLLs and executables in Windows ecosystems. It is standard for Windows NT, 95, and 32 [2,3]. The various sections in the PE file format are the MS_DOS header a.k.a. EXE_header, followed by the DOS_stub, PE_signature, PE_File_header a.k.a. COFF_header, Optional_header, and PE_Sections. The DOS header contains 19 fields. In this header, various machine instructions are included to support the backward compatibility of the system. The first two-byte ASCII value is MZ, which stands for "Mark Zbikowski", the person who developed the DOS linker. Figure 1 shows the Hex dump, showing a sample of all the headers of a PE file [8]. The DOS stub contains an error message that will be printed in case of machine mismatch. The last field of the MS_DOS header, called elfnew, holds a pointer at an offset of 0x3C from the beginning of the MS_DOS header. This pointer points to the PE_File_header. The PE_signature is nothing but an ASCII value "PE" followed by two bytes of NULL characters. Prior to the linking process of a file, it is called an object file; after linking it is called an executable file. The Common Object File Format (COFF) header, a.k.a. File_header, contains some information that is of use to both executable files as well as object files. COFF_header has seven fields, and it exists both in object files as well as in executable files. Optional_header has relevance in executable files but not in object files, hence the name Optional_header. This plays a crucial role in malware detection through ML, as it contains a lot of information regarding linker versions, OS versions, sizes and pointers to code and data, image versions, etc.: a total of 30 attributes. PE_Sections consists of a variable number of sections as specified by the No_of_sections field in the COFF_header. Common types of sections that can be seen in a PE file are text_section or code_section, which carries the executable program, textbss_section, which carries extended text on additional linking, data_section, which carries initialized data, bss_section, which carries uninitialized data, and rsrc_section, which carries resource data.

Several studies in the domain of Windows executable file malware sensing, predominantly based on PE file header attribute analysis, are critically reviewed [9]. Even a tenuous improvement in the detection rate is a great achievement as far as malware detection is concerned. Another focus point at the same time in CTI is optimization without compromise in detection rate. Attribute reduction contributes to time and space optimization which, is of great value for CTI tools. It is also observed that an updated PE file attribute dataset would aid researchers in this domain in developing efficient CTI tools. The contributions of this research include:

- Developing novel swarm-optimization-based PE header attribute-minimizing algorithms.
- Performance evaluation on existing benchmark datasets using various ML classifiers.
- Proposing an updated, pure-PE file header attribute dataset called the SOMLAP dataset (Swarm Optimization and Machine Learning Applied to PE Malware Detection)
- Performance comparison with existing works and validation of the SOMLAP dataset.

The upcoming sections of the article are organized as follows. A critical literature study analyzing the pros and cons of existing methods in the domain of malware detection with a focus on PE files is presented in Section 2. The SOMLAP dataset construction process with details of all the features is discussed in Section 3. The ACO, CSO, and GWO swarm optimization algorithms are discussed in Section 4. Section 5 includes experimental results and discussion, while Section 6 concludes the paper.

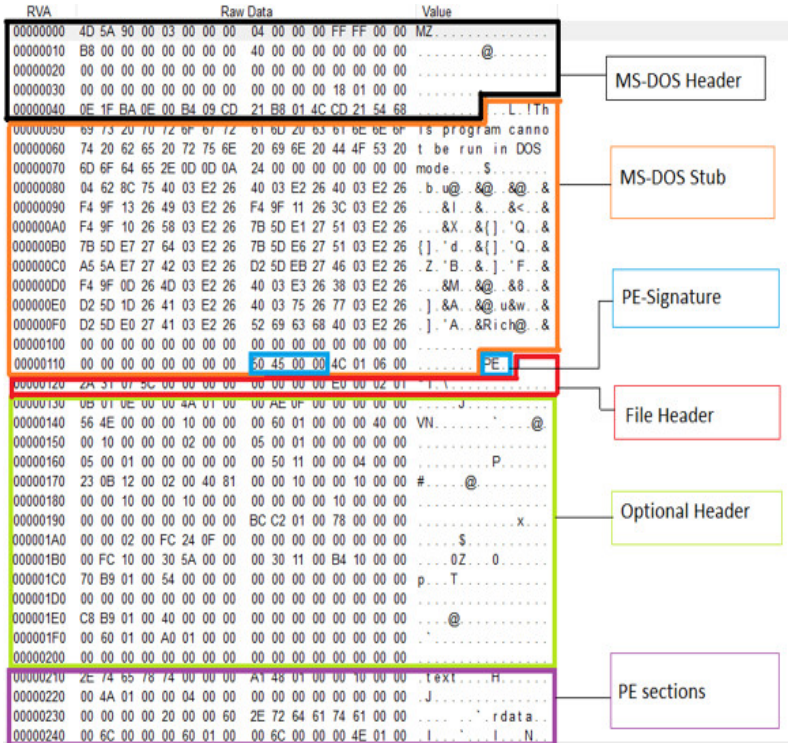


Figure 1. A sample hex dump showing all the headers of the PE file.

2. Literature Review

The ML-based malware detection system developed by Schultz et al. [4] is one of the earliest works in this domain. It is based on the information contained in the PE executables, such as string (both in .exe and non-executables) features and byte sequence features. They used a classification method based on Ripper, naïve Bayes (NB) and multi-NB. The data set consists of 4266 programs with 76% malicious and 24% benign files. The malware data was gathered from various FTP sites and labeled as malicious by commercial AV software. The data source for the benign files was the newly installed Windows 98 machine. They selected plain text strings as the feature in the PE format. An accuracy of 97.76% using multi-naïve Bayes was recorded, and no feature reduction was carried out. In this study, it is proved that the ML approach is more efficient than signature-based methods. A recent survey done by Namita and Prachi [10] concluded that most of the literature on PE malware analysis makes use of ML methods.

Wang et al. [11] discussed a virus detection method for an unknown computer virus using data mining algorithms. They collected 3265 malicious binaries and 1001 benign programs from a database of Columbian University. The feature set is a byte sequence of each instruction of the program. They reduced the feature set based on the information gain frequencies. An accuracy of 91.4% with NB and Decision Tree (DT) classifiers was reported. A very useful and interesting fact revealed in their research is that the percentages of malicious and the benign files in the dataset will affect the result drastically. However, PE header attributes are not considered in malware detection, and there is scope for higher accuracy.

Sung et al. [12] proposed the SAVE (Static Analyzer of Vicious Executables) algorithm. The similarity measure between the system calls, before and after the obfuscation, is considered for differentiating malware and benign files. They used the Euclidian distance to calculate the similarity between the features, with an additional solution that is the Optimal Alignment Algorithm. SAVE accurately detected the given worms and viruses that are presented in the data set, while many commercial AVs failed to do so for a minor obfuscation. However, they did not use PE file header attributes and no feature selection is mentioned.

Kolter and Maloof [13] built the Malicious Executable Classification System (MECS). It detects unknown malicious executables "in the wild", i.e., without removing any obfuscation.

They collected 1971 (54.4%) system and non-system executables (benign files) and 1651 (45.6%) malicious executable files. Benign files are from Source Forge and the malicious files are from VX Heaven's website. The classification techniques used are NB, boosted NB, Support Vector Machines (SVM), boosted SVMs, DT, and boosted DT. They used n-grams as the feature set, which are extracted from the byte sequences. They reduced the feature set to top 500 n-grams based on the pilot studies. They achieved accuracy of 99.6% with boosted DT, which is very efficient. However, fewer malware sources are considered in the dataset. Moskovitch et al. [14] presented a methodology for malware categorization based on concepts from text categorization. The source of 7688 malicious files was Vxheavens, while the benign dataset of 22,735 files was taken from Kaspersky AV. They used four classification algorithms: Artificial Neural Networks (ANN), DT, NB, and SVM. The features are text vocabularies of 5-g. They achieved accuracy of 94.6% with ANN and 5-g data vocabularies. However, this is not based on PE file header attributes and there is also scope for an increase in accuracy.

Elovici et al. [15] presented how eDare performed under five plugins, three ML techniques, and two types of inputs (n-gram and PE executables). They collected a repository of 7694 malicious files and 22,736 benign files. Three classification techniques, ANN, DT, and Bayesian Networks (BN) were used. They used two static features: n-grams and PE headers of win32 executables. Some of the PE attributes are not explored in this work. Feature reduction was done based on the Fischer score, by which they reduced 5500 features to top 3005-grams. With DT, an accuracy of 95.5% was achieved, which has a lot of scope for improvement.

Ye et al. [5] built an IMDS (Intelligent Malware Detection System) that labels unseen Malware files based on PE header data. The data set included 29,580 Windows PE files, of which 12,214 (41.29%) were benign and 17,366 (58.71%) were malicious executables. The data source for the malicious files was the anti-virus laboratory of King Soft corporation, and the source for the benign files was from the Windows 2000/NT and XP operating system. They used the Objective Oriented Association (OOA) mining-based classification method. By using the PE parser, they extracted the Windows API execution calls. The IMDS achieved an accuracy of 93.07%. It is worth noting that the percentage of malicious files is significantly higher than the percentage of the benign files, and the result may tend towards the TP more; results may vary if the percentage of the malicious files is far lower than the benign files [14]. The detection rates are low.

Walenstein et al. [16] experimented on malware detection with and without header information separately and concluded that efficiency is superior with header information. The data set comprised in total 23,906 executables, where 15,641 (65%) are malicious and 8265 (34.6%) are benign files. The source for both the malicious and the benign files was McAfee Threat Center. They used NB, J48, SVM, Random Forest (RF), and IB5 with 10-fold cross validation. The PE header fields are the feature set of the model. They performed feature reduction based on the info gain. They achieved accuracies of 99.1% and 99.8% in multi-class and binary respectively. However, the data set is not purely PE header attributes; it also includes API calls and DLLs. Ye et al. [17] proposed the first paper that uses post-processing techniques of associative classification in malware detection. They upgraded their IMDS system [5] to a CIMDS system. They proposed an effective way, CIDCPF, to detect the malware. Implicitly, CIDCPF adapts several post-processing techniques, including rule pruning, rule ranking, and rule selection. A total of 35,000 malicious files and 15,000 benign files were collected from the antivirus laboratory of Kingsoft Corporation. The detection rate was 76.5174%, which is +40.2653% more than Kaspersky AV. However, there is a significant decrease in the accuracy.

Salehi et al. [18] proposed a dynamic malware detection system based on analyzing API calls and their respective arguments. Multiple classifiers, such as Rotation RF, RF, J48, FT, and NB are used. The data set was categorized into three: API-List, ARG-List, and API-ARG list. The data set included 826 malware files and 385 benign files, comprised of seven categories. They created data sets based on the fact that samples with similar behaviors need to call the same APIs with similar arguments. With Rotation RF, an accuracy of 98.4% was reported. Belaoued and Mazouzi [19] proposed a malware detection system that

categorizes a file in three different phases, which are feature extraction, feature selection, and the decision. A total of 552 malware PE files with 12 different malware categories were collected from Vxheavens. An accuracy of 97.25% with statistical chi-square-based classifiers is reported. The drawback here is that they considered only an optional header in the feature extraction, which limits the accuracy. Akour et al. [20] built an application which connects to different free web-based malware detection systems. These websites are public malware repositories, such as AutoShun, PhishLabs, Kaspersky, StopBadWare, Sophos, and NetCraft. Feature reduction was not done. It was reported that web-based malware detection systems are able to contribute an accuracy of 50%.

Zatloukal and Znoj [21] accomplished two different goals. They tried to find out if detection can be achieved by studying multiple PE headers, for both the virus file and the host file. A total of 9101 samples were gathered to create software that searches multiple PE header files for signatures. A total of 9.884% of malware samples were found to have multiple PE headers and 5.772% benign samples had multiple hosts. In the paper, they also reported that 56.227% are attached to the same application, called "uninstall000.exe". David et al. [22] highlighted the importance of sections in PE files and identified that malicious files have more unnamed sections. They also stressed that if the section names are not default and their length is greater than eight characters, then there is a greater chance that they are malware files. Vidyarthi et al. [23] mixed both static and dynamic analysis for detecting the malware files. They proposed a framework that is used to automate the process of the runtime calculation of the executables. The framework uses text mining to extract significant features for classification of the file. They worked on 180 samples, which contained both benign files and malware, and performed static analysis on sections in the PE header. Feature reduction using information gain on the raw dataset was carried out. The features were reduced to 232 in total. They achieved an accuracy of 92% with SVM classifiers, leaving scope for improvement.

Raff et al. [24] showed that, with minimal domain knowledge, neural networks can produce better accuracies in malware detection. They created a baseline in which they used PortEx to extract 115 features using RF and Extra RF. In the actual experiment, they used neural networks like FC and LSTM. Three datasets were used, among which Group A consists of 175,875 malware samples and 269,431 samples collected from VirusShare and Open Malware. Group B consists of 200,000 malware and benign samples each. Group A achieved an accuracy of 99.5%, and Group B achieved an accuracy of 71.5%. Then they applied neural network algorithms to the dataset, which was the raw byte patterns of the first three headers, and reported an accuracy of 90.8% in Group A and 83.7% in Group B. Zhang [25] proposed a new method for detection of PDF-based attacks, extending their MLP Neural network model [26]. They gathered extensive data, which consists of 105,044 files with 48 features, both for testing and training. The features consist of metadata, PDF structure, object characteristics and more. They achieved an accuracy of 95.12% with the MLPdf model and 93.17% accuracy in the case of MLPdf and principal component analysis (PCA) feature selection. Zhang also stressed that the dimensionality is reduced by 33% and time for learning is significantly reduced by 22%. Maleki et al. [27] mainly concentrated on packed malware. They gathered 971 executable files and extracted 30 features for their dataset. A forward feature selection process reduced the features to eight. With SVM, RF, NN, ID3, and NB classifiers, a best accuracy of 98.26% was reported. In a recent work by Chen et al. [28], PE file malware detection was done by integrating header data, entropy data, opcode n-grams, and API data for the PE files. By applying PCA, they selected 79 attributes. They tried various classifiers to evaluate them; among them, the XGBoost classifier reported the highest accuracy, 99.56%. However, there is scope to reduce the features further.

Kumar et al. [29] contributed to developing a standard benchmark Windows malware dataset called the ClaMP dataset in two different versions, raw and integrated. This model is based on the PE header files, in which they considered only three main headers, which were the DOS header, file header and optional header. The data set consisted of 5180 samples of

both malware and benign files. Malware files were collected from VirusShare and benign files were collected from freshly installed Windows XP and Windows 7. They used multiple classifiers, such as RF, LR, LDA, DT, NB, and KNN. They prepared two datasets, raw and integrated. The raw dataset was extracted with 55 features and the integrated dataset with 68 features. They performed feature reduction based on the assumption that the integrated set outperforms the raw dataset performance. The integrated dataset contained 28 features from the raw dataset along with 26 Boolean features and 14 derived features. They achieved a maximum accuracy of 98.78% for the integrated set with the Random Forest classifier, +1.35% compared to the raw dataset with the same classifier. Few attributes from the header fields were omitted, which needs further investigation. Instead of two different datasets, a unified dataset must be developed. Non-PE header attributes are included in the integrated dataset. Penmatsa et al. [30] applied soft computing methods on the ClaMP dataset. Rough set-based filtering of attributes combined with ACO achieved 97% data reduction with a marginal loss of accuracy.

There is a scope of further research after summarizing the outcome of literature survey. As malware detection systems demand close-to-100% accuracy, there is scope for research to improve the accuracy. There is scope to add several contributing attributes of the PE file header to the existing benchmark datasets [29]. There is a need to develop a new dataset for malware detection with attributes exclusively extracted from PE file headers. There is a need to research efficient swarm-optimization-based feature selection methods to design highly efficient and faster malware detection systems. This work proposes a new dataset called the SOMLAP (Swarm Optimization and Machine Learning Applied to PE Malware Detection) dataset. The goal of our research is to develop a new dataset by enhancing the existing dataset. Applying swarm optimization methods in wrapper mode can further reduce the proposed dataset's attributes.

3. SOMLAP Dataset Construction

This data is inspired by the contributions made by Kumar et al. [29] to the ClaMP dataset and is an extended version. Most of the features from ClaMP are retained, and new features are included and proposed for a single dataset that is based purely on PE file headers. A total of 51,409 samples were extracted. Out of these, 19,809 (38.54%) malware files were gathered from Virus Share [31], and 31,600 (61.46%) benign executables and DLLs were gathered from Windows 10 OS. The "pefile" [32] module was employed for feature extraction from executables and DLLs. Figure 2 is a block diagram showing the process of feature extraction. Attributes from four parts of the PE file were taken, totaling 108. The DOS_header, COFF_header, Optional_header, and various sections are shown in the Tables 1–9.

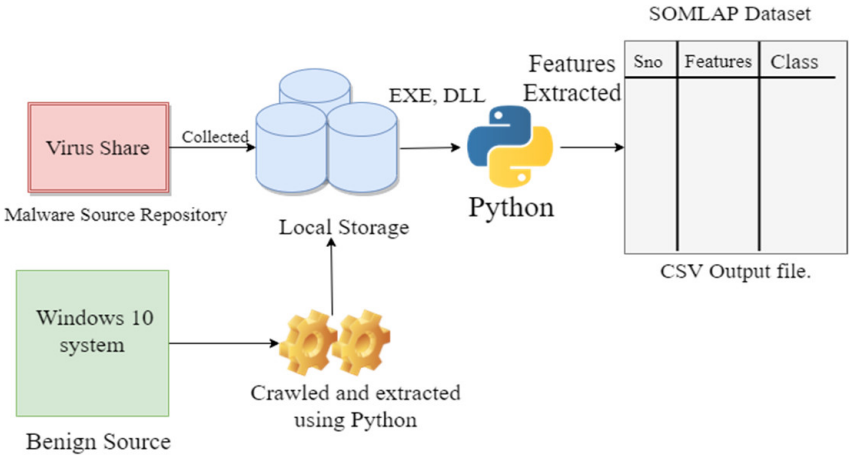


Figure 2. SOMLAP data extraction process.

3.1. Features Taken from DOS_Header

The DOS header has a total of 19 features; among them, only 16 were extracted. The excluded features were e_magic, e_res, and e_res2. The reasons for excluding them are that the e_magic field is used to identify the MS-DOS-compatible file type. All files which are compatible set a constant value to 0x5A4D. The e_res and e_res2 fields are reserved. Hence, they will not contribute to classifying whether a file is malware or not. Table 1 gives the list of DOS_header attributes considered for the SOMLAP dataset. Kumar et al. developed a benchmark dataset from PE headers; however, they proposed two different datasets [29]. The integrated ClaMP dataset is better; however, only six attributes from the DOS_header are taken.

Table 1. DOS_header attributes of SOMLAP.

S. No.	Field	Description
1.	Blp	Last page bytes
2.	Fp	File pages
3.	Rn	Relocation number
4.	Prhdr	Header size in the paragraph
5.	Minpar	Min. paragraphs required in extra
6.	Maxpar	Max. paragraphs required in extra
7.	Ivalss	Initial value of SS
8.	Ivalsp	Initial value of SP
9.	doscksum	Checksum value
10.	Iip	Instructor pointer initial value
11.	Ics	CS Initial value.
12.	Rta	Relocation table file address
13.	Ovn	Number given to overlay
14.	Idoem	ID of the OEM
15.	Infoem	Information of OEM
16.	exehdradr	Exe header address

3.2. Features Gathered from The Coff_Header or File_Header

The file header has a total of seven features, and, in this work, we have extracted all seven features. The Time_Date_Stamp, a 32-bit field, is converted to binary by applying a rule [29]. Malware usually has unusual dates, such as a date before 1980 or beyond the current date. As the first DOS was released in 1980, all the executables should have a date after 1980. If the Time_Date_Stamp lies between 1980 and the current date, it is considered to be benign, or else it is considered as a malware file. Table 2 gives the list of features extracted from COFF_ header.

3.3. Features Extracted from The Optional_Header

The optional header has a total of 30 features. In our dataset, we included all 30 features. A few features are modified (ibase, filealig, and secalign) according to the rules given by the PE format documentation [33]. The characteristics of COFF_header are taken as discrete binary features in the integrated ClaMP dataset; however, here we took it as a single attribute with a hexadecimal value. Table 3 gives the list of extracted attributes from the optional header.

Table 2. Coff_header attributes of SOMLAP.

S. No.	Field	Description
17.	mach	Identifies the machine for which compilation of the file is done
18.	nsec	Section count after the header part
19.	tds	Binary value representing if the time is between 1980 and now or not
20.	ptrst	Symbol table pointer
21.	stcnt	Symbol table entries count
22.	ohs	Optional_header size
23.	char	Characteristics

Table 3. Attributes from Optional_header.

S. No.	Attribute	Description
24.	sig	Signature of the image
25.	majlv	Major linker version
26.	minlv	Minor linker version
27.	codesize	Total size of code in all sections together
28.	initdatsize	Initialized data size
29.	uninitdatsize	Uninitialized data size
30.	adrentpt	Address of entry point
31.	cbase	Code section base pointer
32.	dbase	Data section base pointer
33.	ibase	Image section base pointer
34.	secalign	Section alignments loaded to memory (bytes)
35.	filealign	Image file raw data alignment (bytes)
36.	majosver	The OS's major version
37.	minosver	The OS's minor version
38.	majiver	Image file's major version
39.	miniver	Image file's minor version
40.	majssver	File subsystem's major version
41.	minssver	File subsystem's minor version
42.	win32vv	Reserved. (0 by default)
43.	soi	Image file size with all headers (bytes)
44.	soh	Combined size of all headers.
45.	optcksum	Checksum value of image
46.	ss	The invoked subsystem to run the exe file
47.	dllch	Characteristics of the DLL
48.	sosr	Size of stack reserve
49.	sosc	Size of stack commit
50.	sohr	Size of heap reserve
51.	sohc	Size of heap commit
52.	ldflg	Loader flag (obsolete)
53.	ndirent	The number of directory entries in the optional header.

3.4. Section Table

Optional_header is followed by the section table. The section table contains an array of header structures whose number is indicated by the nsec field of COFF_header. The length of every structure is 40 bytes. Text and data are common sections. There are some special sections which are not present in every executable, viz., bss, cormeta, debug, edata, idata, pdata, rdata, reloc, rsrc, textbss, sbss, sdata, srdata, sxdata, tls, and vsdata. However, the most common sections are text, data, textbss, bss, rsrc and idata [34]. Sections play an important role in malware identification [22]. Therefore, we included these six sections in our dataset. Each section has 10 attributes, and they are the same for all. Entropy of the sections is very crucial for differentiation between benign and malware files [29].

Malware creators will try to evade malware detection from anti-malware tools by showing reduced entropy. Only ML techniques can address this issue. For each section, entropy is calculated and included as a derived additional attribute. Therefore, a total of 11 attributes are there for each section including entropy. However, after measuring the standard deviations of all the attributes, a few are eliminated from the bss and textbss sections. From the bss section only seven attributes and from the textbss section only four attributes are included. Therefore, from the section table, from all of the six sections together, 55 attributes are included in the dataset. The list of the attributes of all of the six sections are given in Tables 4–9. The algorithm for feature extraction is listed as Algorithm 1.

Table 4. The section table header attributes of the text section.

S. No.	Field	Description
54.	Text_mscfaddr	Address of the file.
55.	Text_secsize	Section size that the memory was loaded with.
56.	Text_byteaddr	The loaded section's first byte address.
57.	Text_datsize	Size of the data initialized on the disk.
58.	Text_ptrrawdat	Pointer to the COFF file first page of raw data.
59.	Text_ptrreloc	Pointer to the starting of relocation entries
60.	Text_ptrlinenum	Pointer to line numbers.
61.	Text_numrelocs	The sections count of entries for relocations.
62.	Text_numlinenums	Count of entries for line numbers.
63.	Text_char	Image characteristics.
64.	Text_entro (A derived attribute)	This is a derived attribute, and not a part of the section. This is the entropy calculated for the section.

Table 5. The section table header attributes of the idata section.

S. No.	Field	Description
65.	Idata_mscfaddr	Address of the file.
66.	Idata_secsize	Section size that the memory was loaded with.
67.	Idata_byteaddr	The loaded section's first byte address.
68.	Idata_datsize	Size of the data initialized on the disk.
69.	Idata_ptrrawdat	Pointer to the COFF file first page of raw data.
70.	Idata_ptrreloc	Pointer to the starting of relocation entries
71.	Idata_ptrlinenum	Pointer to line numbers.
72.	Idata_numrelocs	The section count of entries for relocations.
73.	Idata_numlinenums	Count of entries for line numbers.
74.	Idata_char	Image characteristics.
75.	Idata_entro (A derived attribute)	This is a derived attribute, and not a part of the section. This is the entropy calculated for the section.

Table 6. The section table header attributes of the rsrc section.

S. No.	Field	Description
76.	Rsrc_mscfaddr	Address of the file.
77.	Rsrc_secsize	Section size that the memory was loaded with.
78.	Rsrc_byteaddr	The loaded section's first byte address.
79.	Rsrc_datsize	Size of the data initialized on the disk.
80.	Rsrc_ptrrawdat	Pointer to the COFF file first page of raw data.
81.	Rsrc_ptrreloc	Pointer to the starting of relocation entries
82.	Rsrc_ptrlinenum	Pointer to line numbers.
83.	Rsrc_numrelocs	The section count of entries for relocations.
84.	Rsrc_numlinenums	Count of entries for line numbers.
85.	Rsrc_char	Image characteristics.
86.	Rsrc_entro (A derived attribute)	This is a derived attribute, and not a part of the section. This is the entropy calculated for the section.

Table 7. The section table header attributes of the data section.

S. No.	Field	Description
87.	Data_mscfaddr	Address of the file.
88.	Data_secsize	Section size that the memory was loaded with.
89.	Data_byteaddr	The loaded section's first byte address.
90.	Data_datsize	Size of the data initialized on the disk.
91.	Data_ptrrawdat	Pointer to the COFF file first page of raw data.
92.	Data_ptrreloc	Pointer to the starting of relocation entries
93.	Data_ptrlinenum	Pointer to line numbers.
94.	Data_numrelocs	The section count of entries for relocations.
95.	Data_numlinenums	Count of entries for line numbers.
96.	Data_char	Image characteristics.
97.	Data_entro (A derived attribute)	This is a derived attribute, and not a part of the section. This is the entropy calculated for the section.

Table 8. The section table header attributes of the bss section.

S. No.	Field	Description
98.	bss_phyaddr	Address of the file.
99.	bss_virsize	Size of the memory loaded with section.
100	bss_viraddr	The loaded section's first byte address.
101.	bss_datsize	Size of the data initialized on the disk.
102.	bss_ptrrawdat	Pointer to the COFF file first page of raw data.
103.	bss_char	Image characteristics.
104.	bss_entro (A derived attribute)	This is a derived attribute, and not a part of the section. This is the entropy calculated for the section.

Table 9. The section table header attributes of the textbss section.

S. No.	Field	Description
105.	bss_phyaddr	Address of the file.
106	bss_virsize	Size of the memory loaded with section.
107	bss_viraddr	The loaded section's first byte address.
108.	bss_char	Image characteristics.

Algorithm 1. Feature extraction from the malware and benign executable files

Input: The set of malware and benign executables

Output: Dataset with 108 features

```
procedure FeatureSetGeneration 𝔘, μ
dataset [ ] ↩ρ 0
for p in 𝔘 do ⊙ Extracting features from Malware
class ↩ρ 1
virus [ ] ↩ρ 0
fetch_dos_header(p, virus)
fetch_file_header(p, virus)
fetch_optional_header(p, virus)
fetch_pe_header(p, virus)
dataset [ ] ↩ρ virus [ ]
for p in μ do ⊙ Extracting features from benign
class ↩ρ 0
Virus [ ] ↩ρ 0
fetch_dos_header(p, virus)
fetch_file_header(p, virus)
fetch_optional_header(p, virus)
fetch_pe_section(p, virus)
dataset [ ] ↩ρ virus [ ]
return dataset [ ]
procedure fetch_dos_header ρ, ψ
for k in ρ do
if k ≢ ω where ω = {fields} do
ψ ↩ρ k
return ρ, ψ
procedure fetch_file_header ρ, ψ
for k in ρ do
if k ≡ τ where τ ={timeStamp} do
date ↩ρ Convert TimeDateStamp into Human readable Format
⊙ 333936000 is the date: 1 August 1980 12.00 hrs and 1574157742 is the date: 19 November 2019
If date ≥ 343476142 and date ≤ 1574157742 do
Virus [date] ↩ρ 0
else do
Virus [date] ↩ρ 1
else do
ψ ↩ρ k
return ρ, ψ
procedure fetch_optional_header ρ, ψ
for k in ρ do
if k ≡ ν where ν = {ImageBase} do
If k.value ≡ Z where Z = {principles for ImageBase}
Virus [imageBase] ↩ρ 0
else
Virus [imageBase] ↩ρ 1
else if k ≡ ν where ν = {SectionAlignment} do
If k.value ≡ Z where Z = {principles for SectionAlignment}
Virus [SectionAlignment] ↩ρ 0
else
Virus [SectionAlignemnt] ↩ρ 1
else if k ≡ ν where ν = {FileAlignement} do
If k.value ≡ Z where Z = {principles for FileAlignement}
Virus [fileAlignment] ↩ρ 0
else
Virus [fileAlignment] ↩ρ 1
else do
ψ ↩ρ k
return ρ, ψ
procedure fetch_ text pe_section ρ, ψ
for k in ρ do
if k ≡ ℬ where ℬ = {"", "code","data" ... } ⊙ include set do
ψ ↩ρ k
return ρ, ψ
```

The summarized enhancements made to the proposed SOMLAP dataset are:

- An extension over the benchmark dataset [29] to research for a possible improvement in malware detection accuracy;
- Increased number of samples, for a total of 51,409;
- Updated malware sources from Virus Share and benign sources from Windows 10;
- An increased attribute size to 108, including several features from six sections of
- the section tables;

- The dataset attributes are pure PE-header-based, which is meant to prove the capability of PE header fields in efficient malware detection.

4. Significant Feature Selection Using Swarm Optimization

One of the goals of this research was to identify significant features and thereby contribute to designing highly efficient and fast malware detection systems. The 108 PE header features were chosen intuitively and need further experimentation to extract the most significant features maintaining full feature accuracy. An efficient feature reduction process can reduce the dimensionality of the data that are redundant or noisy. These unnecessary features give inconsistent results while using such data sets in decision-making using machine learning [35]. Reducing the number of features or removing these unnecessary features will improve the performance of machine learning classifiers in terms of both accuracy and time and also eliminate any chances of getting overfitted models by avoiding the curse of dimensionality [36].

Feature selection approaches are generally categorized into four categories: filter-based, wrapper-based, embedded-based, and hybrid [37]. Wrapper-based approaches are widely used nowadays in machine learning. In this approach, the learning algorithm itself is also used for feature selection. A subset of features are selected based on the accuracy of the learning algorithm [38]. There are various wrapper-based feature selection approaches, designed using different bio-inspired meta-heuristic optimization techniques, viz., Ant Colony Optimization (ACO), Cuckoo Search Optimization (CSO), Fire-Fly Optimization (FFO), Grey Wolf Optimization (GWO), and many more. In this work, for feature optimization, we have chosen three algorithms, ACO, CSO, and GWO. Algorithms from different timelines of history and with different numbers of tuning parameters were chosen. ACO has six, CSO has four, and GWO has just one tuning parameter. All three algorithms have different run times, with ACO the slowest and GWO the fastest.

In wrapper-based approaches, one extensively used evaluation metric is the fitness of the selected feature subset, which is dependent on the classification accuracy of the ML model and the number of features selected, as shown in Equation (1), where F represents the total feature set of the given dataset, and F_s represents the features subset. $Accuracy(F_s)$ represents the classification accuracy of the machine learning model using the selected feature subset F_s, ω is a constant that is used for tuning the fitness function, and $|F|$ represents the number of features in the feature set F.

$$Fitness(F_s) = \omega * Accuracy(F_s) + (1 - \omega) * \left(\frac{|F| - |F_s|}{|F|} \right) \quad (1)$$

4.1. Ant Colony Optimization for Feature Selection

Dorigo first introduced ACO in 1992, intending to find the optimal path in a graph [39]. The approach is based on natural ant behavior in the process of searching for food. Dorigo observed that the ants randomly move while searching for food, and, once an ant finds the food, all ants will converge towards the shortest path between the nest and the food. The main reason for this convergence is a chemical called pheromone, which is deposited by the ants along the path that they travel. Dorigo generated the behavior of these ants artificially to find the optimal path in a graph. Later, many researchers developed ACO further and used it in solving various hard combinatorial problems, like traveling salesman problems [40], job scheduling problems [41], and including feature selection problems [42,43].

A simple ACO has five steps: initialization, ant solution construction, evaluation of each solution, updating the best solution, and pheromone updating. Algorithm 2 details the process of ACO wrapper feature selection. In the initialization phase of the algorithm, the ACO parameters α, β, ρ, q, the initial pheromone concentration ph_0 on each branch, and the fitness tuning parameter ω are initialized. In each iteration, every ant randomly starts at one feature and will select a subset of features until there is no improvement in the fitness of that ant (step-3 to step-7) using Equation (2). When an ant selects a feature, the corresponding

subset of features is evaluated using a fitness function, Equation (1) (step-8 to step-11), and the best solution will be updated with an ant solution that has maximum fitness (step-9). Subsequently, pheromone will be deposited by all ants (step-13 to step-15), and, also, a portion of pheromone on each branch will be evaporated (step-16 and step-17), as shown in Equations (4) and (5). Once all the iterations are over, the best solution, i.e., the selected feature subset with maximum fitness, will be returned (step-18).

$$P_i^j = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_k \tau_{ik}^{\alpha} * \eta_{ik}^{\beta}} \tag{2}$$

where $j, k \in \{F - F_i\}$, and τ_{ij} is the average pheromone concentration on the branches between the features that are already selected by ant i, i.e., F_i and the next probable feature f_i^j, and η_{ij} represents heuristic information gain by the feature f_i^j, which is the difference in fitness with and without this feature, as shown in Equation (3). Parameters α and β are the weights for tuning pheromone concentration and the heuristic respectively.

$$\eta_{ij} = Fittness\left(F_i \parallel f_i^j\right) - Fittness(F_i) \tag{3}$$

$$\tau_{xy}(t+1) = (1-\rho) * \tau_{xy}(t) + \sum_{a_i} \Delta_{xy}^i \tag{4}$$

where

$$\Delta_{xy}^i = \begin{cases} \frac{q}{|F_i|} & if\ x, y \in F_i \\ 0 & otherwise \end{cases} \tag{5}$$

Algorithm 2. Ant Colony Optimization Wrapper Feature Selection Algorithm

Input: Labeled training D_T and evaluating D_E dataset, number of features n_F, number of ants n_a, number of iterations n_T, initial pheromone concentration ph_0, and tuning parameters α, β, ρ, q, ω

Output: A subset F_s of feature set F that gives the maximum fitness over D_E.

Auxiliary: Pheromone Matrix $ph = \{\tau_{xy} = ph_0 \forall\ x, y \in F\}$, ant solutions $\{F_1, F_2, F_3, \ldots\ldots, F_{na}\}$, where $F_i = \varnothing$, ant fitness $\{Ft_1, Ft_2, Ft_3, \ldots\ldots, Ft_{na}\}$, and the best solution $F_s = \varnothing$.

1. For each iteration t $(t = 1, 2, 3, \ldots.., n_T)$ do
2. For each ant a_i $(\forall\ i = 1, 2, 3 \ldots\ldots, n_a) do$
3. Choose a feature f_i^1 randomly from the feature set F and add it to F_i;
4. While more features can be added to F_i do
5. For each feature f_i^j in $\{F - F_i\}$ do
6. Calculate probability P_i^j using Equation (2);
7. Choose the next feature f_i^j from the feature set $\{F - F_i\}$ with max P_i^j and add it to F_i;
8. Generate $D_T{}'$ and $D_E{}'$ from D_T and D_E for features selected F_i by ant a_i
9. Train the ML model using $D_T{}'$
10. Evaluate the ML model using $D_E{}'$
11. Calculate the fitness of ant using Equation (1) and update Ft_i.
12. Update the best solution F_s with the solution of an ant with maximum fitness.
13. For each ant a_i $(\forall\ i = 1, 2, 3 \ldots\ldots, n_a) do$
14. For each pair of features (f_i^x, f_i^y) $(\forall\ (f_i^x, f_i^y) \in (F_i, F_i))$ do
15. Deposit pheromone on branch xy.
16. For each branch xy $(x, y \in F)$ do
17. Update pheromone τ_{xy} after evaporation
18. Return F_s as best solution

4.2. Cuckoo Search Optimization for Feature Selection

Inspired by the brood parasitism of a cuckoo species, Yang and Deb developed a search algorithm mimicking the biological cuckoo bird in 2009 [44]. Later, it was combined with Levy flight to solve combinatorial problems. Cuckoo Search Optimization (CSO) was used by many researchers for feature selection. Aziz and Hassanien combined CSO with the rough-set theory to create a filter-based feature selection algorithm [45]. A hybrid CSO rough set algorithm was proposed in [46] for feature selection. The main drawback of CSO

is its dependency on the random search for global optimum solutions iterating through solution space [47].

Algorithm 3. Binary Cuckoo Search Optimization Wrapper Feature Selection Algorithm.

Input: Labeled training D_T and evaluating D_E dataset, number of features n_F, number of nests n_c, number of iterations n_T, and tuning parameters α, ρ, σ, λ, ω

Output: A subset F_s of feature set F that gives the maximum fitness over D_E.

Auxiliary: Nests, i.e., solutions $NS = \{N_1, N_2, N_3, \ldots\ldots, N_{n_c}\}$ where $N_i = \varnothing$, a binary representation of features F, the best solution $F_s = \varnothing$, and the fitness of nests $FT = \{Ft_1, Ft_2, Ft_3, \ldots\ldots, Ft_{n_c}\}$

1. For each nest N_i $(\forall\, i = 1, 2, 3 \ldots\ldots, n_c)do$
2. For each feature j $(\forall\, i = 1, 2, 3 \ldots\ldots, n_F)do$
3. $N_i \leftarrow N_i \parallel Random\{0, 1\}$
4. Generate $D_T{}'$ and $D_E{}'$ from D_T and D_E for features selected N_i
5. Train the ML model using $D_T{}'$
6. Evaluate the ML model using $D_E{}'$
7. Calculate the fitness of ant using Equation (1) and update Ft_i.
8. For each iteration t $(t = 1, 2, 3, \ldots.., n_T)$ do
9. For each nest N_i $(\forall\, i = 1, 2, 3 \ldots\ldots, n_c)do$
10. Generate a new solution $\hat{N}_i$ by using Levy flight on N_i using Equation (8).
11. Generate $D_T{}'$ and $D_E{}'$ from D_T and D_E for features selected $\hat{N}_i$
12. Train the ML model using $D_T{}'$
13. Evaluate the ML model using $D_E{}'$
14. Calculate the fitness $\hat{Ft}_i$ of ant using Equation (1)
15. Randomly choose a nest N_j from NS-$\{N_i\}$
16. If $Ft_j < \hat{Ft}_i$, then
17. $N_j \leftarrow \hat{N}_i$
18. $Ft_j \leftarrow \hat{Ft}_i$
19. If $max(FT) > Fittness(F_s)$, then
20. Update the best solution F_s with the solution of the nest with maximum fitness
21. Abandon a portion ρ of nests and generate new nests using Equation (9).
22. Return F_s as best solution

A cuckoo search algorithm is not directly suitable for feature selection; instead, a modified version of a cuckoo search algorithm called a binary cuckoo search algorithm is used for feature selection [45]. In the binary cuckoo search approach, a feature space with n features will be considered as an n-dimensional space with each feature at the corner of that space. The position in a traditional CSO is represented with continuous values, whereas, in binary CSO, the positions are represented by a binary vector of 1s and 0s, with 1 corresponding to selection of the corresponding feature and 0 meaning not selected. Hence, once a new solution is generated by Levy flight, it will not be a binary vector; instead, it will be continuous-valued. The continuous-valued vector will be converted into a binary vector using a sigmoid function, shown in Equations (6) and (7).

$$S\left(N_i^j\right) = \frac{1}{1 + e^{-N_i^j(t)}} \tag{6}$$

$$N_i^j(t+1) = \begin{cases} 1 & S\left(N_i^j\right) > \sigma \\ 0 & otherwise \end{cases} \tag{7}$$

where $\sigma \sim N(0, 1)$ and $N_i^j(t)$ denotes the new value of nest i at dimension, i.e., feature j after generating a new solution at any point.

The binary CSO has four phases: initialization, traversal using Levy flight, evaluation of solutions and updating the best solution, and new nest construction by abandoning worst nests, as shown in Algorithm 3. During initialization, a random binary vector will be generated and evaluated for each nest (step-1 to step-7). Then the remaining steps will run for multiple iterations (step-8). In each iteration, for each nest (step-9), a new solution will be generated using Levy flight using Equation (8) (step-10). Then the solution will be evaluated using an ML algorithm (step-11 to step-13), and the fitness will be calculated using Equation (1) (step-14). The newly generated solution will be compared with a

randomly chosen nest, and that nest may be updated with a new solution (step-15 to step-18). Then the best solutions will be compared and may be updated with the nest with the best fitness (step-19 and step-20). The final step of an iteration is abandoning a portion of the worst nest and generating new nests using Equation (9) (step-21). Then, once the predefined number of iterations is over, the best solution will be returned (step-22).

$$N_i^{t+1} = N_i^t + \alpha * Levy(\lambda) \tag{8}$$

$$N_i^{t+1} = N_i^t + \delta * \left(N_x^t - N_y^t\right) \tag{9}$$

where α is the step size, $Levy(\lambda)$ is the Levy distribution function, δ is a random number between 0 and 1, and N_x^t and N_y^t are two randomly chosen nests from NS.

4.3. Grey Wolf Optimization

Grey Wolf Optimization (GWO) was proposed by Mirjalili et al. [48], mimicking the leadership hierarchy and behavior of wolves while hunting. A grey wolf pack normally contains five to twelve wolves headed by a single alpha wolf, the leader that is responsible for decision making. The beta wolves help the alpha wolf in making decisions and leading the pack. The delta wolves will engage in different activities, like scouting, hunting, and caretaking of the weak. In GWO, the position of the prey, which is unknown, will be considered the best solution. The objective of GWO is to converge all wolves towards the prey by moving relatively with respect to alpha, beta, and delta wolves.

Like CSO, traditional GWO is also not suitable for feature selection. A binary GWO has been used by researchers in feature selection [49,50]. In GWO, all the wolves move in accordance with the positions of alpha, beta, and delta wolves. The continuous-valued positions will be converted back to binary vectors using Equations (6) and (7).

Algorithm 4. Grey Wolf Optimization Wrapper Feature Selection Algorithm.

Input: D_T Training data with labels, D_E Dataset for evaluation, n_F the feature count, n_T iteration count, and tuning parameters ω.

Output: A subset F_s of feature set F that gives the maximum fitness over D_E.

Auxiliary: Wolves, i.e., solutions $W = \{W_1, W_2, W_3, \ldots\ldots, W_{n_w}\}$ where $W_i = \varnothing$, a binary representation of features F, alpha wolf W_α, beta wolf W_β, delta wolf W_δ, the best solution $F_s = \varnothing$, the fitness of wolves $FT = \{Ft_1, Ft_2, Ft_3, \ldots\ldots, Ft_{n_w}\}$, a, $\vec{A}$, $\vec{C}$

1. For each wolf W_i $(\forall\, i = 1, 2, 3 \ldots\ldots, n_w)do$
2. For each feature j $(\forall\, i = 1, 2, 3 \ldots\ldots, n_F)do$
3. $W_i \leftarrow W_i \parallel Random\{0, 1\}$
4. Generate $D_T{}'$ and $D_E{}'$ from D_T and D_E for features selected N_i
5. Train the ML model using $D_T{}'$
6. Evaluate the ML model using $D_E{}'$
7. Calculate the fitness of ant using Equation (1) and update Ft_i.
8. Rank the solutions based on the fitness
9. Identify 1st, 2nd, and 3rd best solutions and update W_α, W_β, and W_δ correspondingly.
10. For each iteration t $(t = 1, 2, 3, \ldots.., n_T)$ do
11. For each wolf W_i $(\forall\, i = 1, 2, 3 \ldots\ldots, n_w)do$
12. Generate the new position of wolf i by moving W_i using Equation (10)
13. Generate $D_T{}'$ and $D_E{}'$ from D_T and D_E for features selected W_i
14. Train the ML model using $D_T{}'$
15. Evaluate the ML model using $D_E{}'$
16. Calculate and update the fitness Ft_i of wolf i using Equation (1)
17. Update W_α, W_β, and W_δ correspondingly
18. Update a, $\vec{A}$, $\vec{C}$
19. If $Fittness(W_\alpha) > Fittness(F_s)$, then
20. $F_s \leftarrow W_\alpha$
21. Return the best solution F_s

The binary GWO has four phases: initialization; identification of alpha, beta, and delta wolves; moving wolves to new positions and evaluating the new position; and updating alpha, beta, and delta wolves, as shown in Algorithm 4. In phase one, i.e., initialization,

all the wolves will be initialized at random positions, the positions will be evaluated using the fitness function shown in Equation (1), and alpha, beta, and delta wolves will be identified (step-1 to step-9). Once all the wolves are initialized, the remaining three phases will happen in multiple iterations (step-10). In each iteration, first, every wolf will be moved with respect to the positions of alpha, beta, and delta wolves using Equation (10) to Equation (14) (step-11 and step-12). The new positions of all wolves will then be evaluated using Equation (1), and the fitness vector will be updated (step-13 to step-16). The positions of alpha, beta, and delta wolves will be updated (step-17) with the new first, second, and third positions. The parameters a, $\vec{A}$, $\vec{C}$ will be updated (step-18). The best solution will then be compared with the solution of alpha and updated (step-19 and step-20). Finally, once all the iterations are over, the best solution will be returned (step-21).

$$W_i(t+1) = Cross\_over_{W_i}\left(W_{\hat{\alpha}}, W_{\hat{\beta}}, W_{\hat{\delta}}\right) \tag{10}$$

where

$$Cross\_over_{W_i}\left(W_{\hat{\alpha}}, W_{\hat{\beta}}, W_{\hat{\delta}}\right) = \begin{cases} W_{\hat{\alpha}} & if\ rand < 0.45 \\ W_{\hat{\beta}} & if\ 0.45 \leq rand < 0.8 \\ W_{\hat{\delta}} & if\ rand \geq 0.8 \end{cases} \tag{11}$$

$$W_{\hat{\alpha}} = \begin{cases} 1 & if\ W_i + cstep_{\alpha,W_i} - 0.5 \geq rand \\ 0 & if\ W_i + cstep_{\alpha,W_i} - 0.5 < rand \end{cases} \tag{12}$$

$$W_{\hat{\beta}} = \begin{cases} 1 & if\ W_i + cstep_{\beta,W_i} - 0.5 \geq rand \\ 0 & if\ W_i + cstep_{\beta,W_i} - 0.5 < rand \end{cases} \tag{13}$$

$$W_{\hat{\delta}} = \begin{cases} 1 & if\ W_i + cstep_{\delta,W_i} - 0.5 \geq rand \\ 0 & if\ W_i + cstep_{\delta,W_i} - 0.5 < rand \end{cases} \tag{14}$$

The *csteps* of any wolf or wolves $\{W_\alpha, W_\beta, W_\delta\}$ can be calculated using Equations (15) and (16).

$$cstep_{w,W_i} = \frac{1}{1 + e^{-(A * D_{w,W_i})}} \tag{15}$$

$$D_{w,W_i} = (C * w) - W_i \tag{16}$$

The vector $\vec{A}$ is a random vector calculated using Equation (17), which is dependent partially on constant a. The vector $\vec{C}$ is a vector of random values derived using Equation (18). Constant a is initialized with a value of 2 and will be reduced in every iteration using Equation (19).

$$\vec{A} = Vector_{n\_att}(2 * a * rand - a) \tag{17}$$

$$\vec{C} = Vector_{n_{att}}(2 * rand) \tag{18}$$

$$a = 2 - itr * \frac{2}{no_{itr}} \tag{19}$$

5. Results and Discussion

5.1. Free Parameters and Classifier Selection

The purpose of the experiments conducted is two-fold: to evaluate the proposed SOMLAP dataset for accuracy and to identify the essential PE file header features and evaluate their performance. All the experiments were carried over a laptop with an Intel i5 8th Gen processor with RAM of 16GB on the Windows 10 platform. The coding was done with Python, and, for evaluating the proposed ACO, CSO, and GWO FS algorithms and the dataset, classifiers available in the Scikit learn library were used. The constants (free parameters) used in all the three optimization algorithms are shown in Table 10.

Table 10. ACL, CSO, and GWO free parameters for all the experiments.

ACO	CSO	GWO
omega = 0.95 alpha = 0.5 beta = 0.3 rho = 0.1 q = 0.5 epsilon = 0.001 no_ants = 10 no_iter = 10	omega = 0.95 alpha = −0.5 pa = 0.25 lam = 0.01 no_nests = 10 no_iter = 10	omega = 0.95 no_wolfs = 10 no_iter = 10

The SOMLAP dataset was already explained in detail in Section 3. It consists of 51,409 samples and 108 features. First, a trail run was conducted to choose a suitable classifier among six classifiers. To do so, the ACO wrapper FS algorithm was run with ten ants and two iterations on the SOMLAP dataset. The results are tabulated in Table 11. From these primary experimental results, it was decided to use a decision tree (DT) classifier in the rest of the experiments. Even though the random forest (RF) method demonstrated higher accuracy, DT is competitive with RF with respect to accuracy and 96% faster.

Table 11. Initial results to choose a suitable classifier.

Exp. No.	Classifier	Accuracy		Sol Length	Time Taken (in Sec) (10 Ants 2 Iteration)
		Full Features	Reduced Features		
1	KNN	98.25%	99.05%	18	262,380
2	Nearest Centroid (NC)	71.25%	89.14%	14	154.3
3	Random Forest (RF)	99.37%	99.40%	17	141,140
4	Gaussian NB (GNB)	61.73%	95.49%	15	358
5	SVM	89.93%	94.11%	18	6430
6	**Decision Tree (DT)**	**99.09%**	**99.31%**	**13**	**4365**

ROC curves give a pictorial view of the performance of the proposed algorithms. The ROC curves for all the classifiers used in this experiment, comparing with and without feature selection, are shown in Figure 3a–f. The ROC curves clearly depict the performance and truthfulness of the ACO wrapper feature selection, by showing the area under the curve (AUC). The AUC, after bio-inspired wrapper feature reduction, is higher than with the full dataset, which proves the algorithm's efficiency.

5.2. Evaluation of the ACO-DT, CSO-DT, and GWO-DT Wrappers on the Benchmark ClaMP Dataset

First, the ACO-DT, CSO-DT, and GWO-DT wrappers for feature selection were evaluated on the ClaMP Integrated Dataset [29], which is the most recent benchmark dataset developed by Kumar et al. The results of five different runs and their averages are shown in Table 12.

All the three Swarm optimization algorithms with DT classifier wrappers were run with equal number of agents and iterations for a fair comparison. Due to randomness in the FS algorithms, and to get a better picture for performance comparison, five runs were considered. From Table 12, for the ClaMP dataset, the ACO-DT wrapper selected fewer features compared with CSO-DT and GWO-DT. The ClaMP dataset has 69 features in total. The best accuracy, as recorded in Table 12, was from ACO-DT's first run, where 18 features were selected with 98.016% accuracy. The best performance with respect to accuracy for CSO-DT was in Run 3, where 39 features were selected with 97.696% accuracy. The best run for GWO-DT, with 19 significant features and 97.248% accuracy post reduction, was recorded in Run 3. Clearly, ACO-DT outperformed in two ways, with fewer features

selected and the highest accuracy. However, as far as time complexity is concerned, GWO-DT was the fastest, followed by CSO-DT, and the slowest one was ACO-DT. Table 13 details the list of features identified in Run 1 of Table 12 for ACO-DT, CSO-DT, and GWO-DT. Table 13 serves as a proof of concept for the selected features. In Table 13, the feature numbers selected from the ClaMP dataset for each header category are detailed. If we look at the average results of the three optimizing algorithms in Table 12, ACO-DT outperformed with respect to detection accuracy, but the run time was very high at 2325 s. CSO-DT is very close to ACO-DT with respect to detection accuracy, and it has an advantage of faster run times at 2.2 s. GWO-DT was the fastest, with an average run time of approximately 1 s, but the accuracies are low compared to ACO-DT and GWO-DT. Therefore, to summarize the results of Table 12, there is always a three-dimensional trade-off in the choice of the best algorithm among ACO-DT, CSO-DT, and GWO-DT, viz., features reduced, accuracy, and time complexity. Since the goal of the research was to identify an optimal feature set with the maximum possible accuracy, ACO-DT has won the race.

Since the swarm optimization algorithms chosen for this study were proved to be promising compared to the existing benchmark dataset, they were also evaluated for the proposed SOMLAP data set.

5.3. Evaluation of the ACO-DT, CSO-DT, and GWO-DT Wrappers on the Proposed SOMLAP Dataset

The ACO-DT, CSO-DT, and GWO-DT optimization–classifier hybrids proved to be very effective for the ClaMP benchmark data sets. In this experiment, the enhanced and proposed SOMLAP dataset was evaluated with the same optimization–classifier combo. The results are tabulated in Table 14.

Table 12. Evaluation of ACO-DT, CSO-DT, and GWO-DT on the benchmark ClaMP dataset.

Run No.	DT Wrapper FS Algorithm	Accuracy with Full Features	No. of Features Selected (out of 69)	Run Time (Sec)	Accuracy after Feature Selection	Iteration Number Where the Best Solution Found
1	ACO-DT	97.50%	18	2093.68	98.016%	7
	CSO-DT		26	2.141	97.312%	10
	GWO-DT		37	1.022	96.929%	3
2	ACO-DT	97.24%	19	2127.89	97.184%	3
	CSO-DT		29	2.256	96.865%	6
	GWO-DT		19	1.05	96.673%	9
3	ACO-DT	97.185%	24	2216.10	97.696%	1
	CSO-DT		39	2.198	97.696%	10
	GWO-DT		24	1.046	97.248%	4
4	ACO-DT	98.144%	21	2354.46	97.824%	4
	CSO-DT		25	2.235	97.502%	2
	GWO-DT		19	1.099	96.609%	6
5	ACO-DT	96.993%	20	2834.20	97.760%	3
	CSO-DT		20	2.259	97.057%	10
	GWO-DT		19	0.960	95.841%	6
Avg.	**ACO-DT**	97.528%	20.4	2325.66	97.696%	3.6
	CSO-DT		27.8	2.2178	97.286%	7.6
	GWO-DT		23.6	1.035	91.261%	5.6

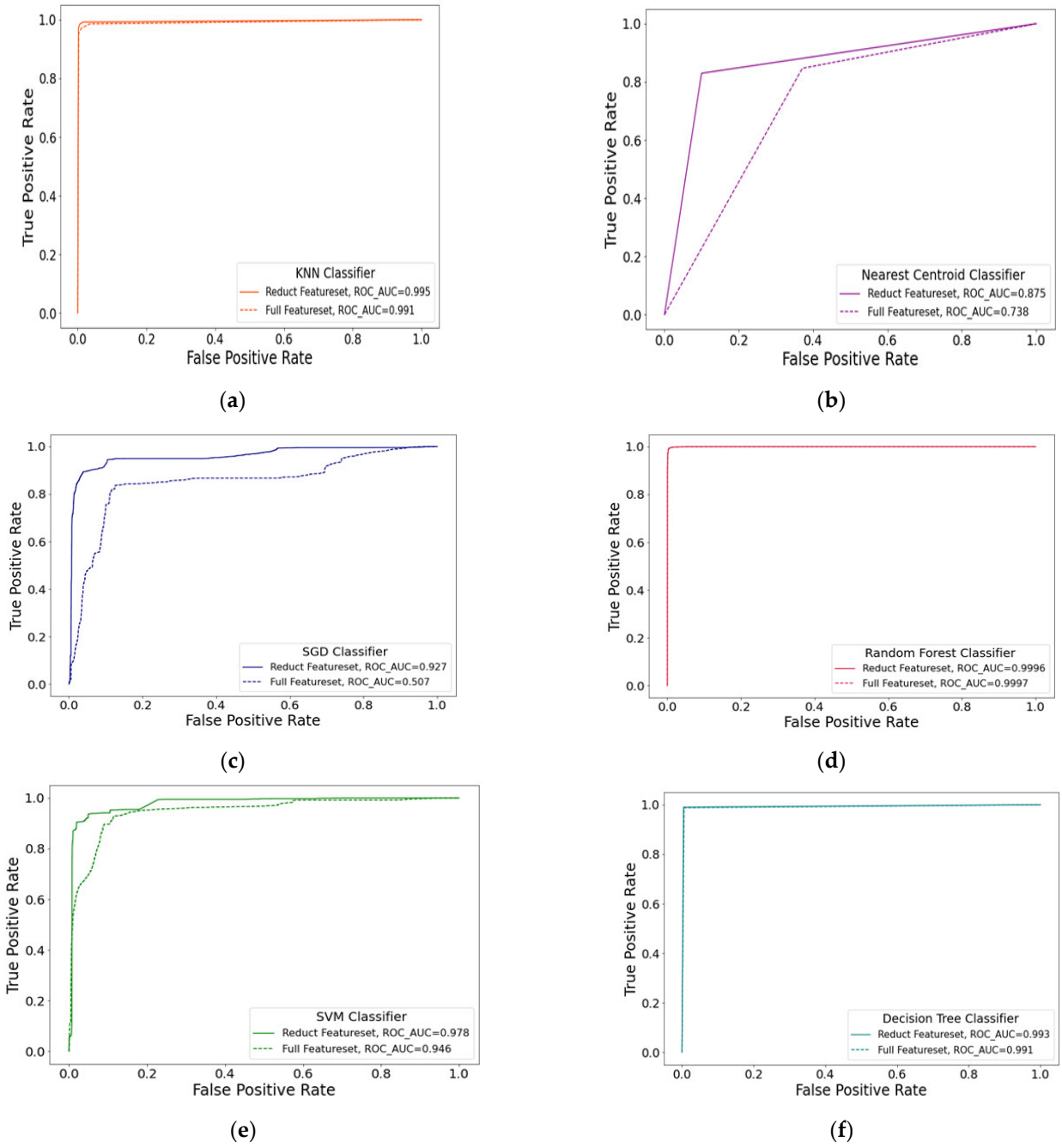


Figure 3. (**a**) ROC curves for ACO-KNN; (**b**) ROC curves for ACO-NC; (**c**) ROC curves for ACO-GNB; (**d**) ROC curves for ACO-RF; (**e**) ROC curves for ACO-SVM; (**f**) ROC curves for ACO-DT.

Table 13. Details of the features selected from Run 1 of Table 12.

FS Algorithm	No. of Features Selected (With Best Accuracy)	Features Selected from DOS_Header (6 Features, 1–6)	Features Selected from COFF_Header (17 Features, 7–23)	Features Selected from Optional_Header (37 Features, 24–60)	Features Selected from Other_Fields (9 Features, 61–69)
ACO-DT	18	1, 2, 3, 6.	7, 8, 12, 13, 16, 17, 21.	28, 30, 32, 37, 43, 44, 45.	NIL
CSO-DT	26	1, 5.	7, 8, 9, 10, 11, 13, 17, 18, 21, 23.	29, 34, 38, 43, 44, 47, 52, 55, 56, 57, 60.	62, 63, 66.
GWO-DT	37	1, 2.	8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 21.	24, 27, 29, 30, 31, 32, 34, 35, 36, 37, 39, 43, 46, 47, 52, 59, 60.	62, 63, 64, 66, 68, 69.

Table 14. Evaluation of ACO-DT, CSO-DT, and GWO-DT on the proposed SOMLAP dataset.

Run No.	DT Wrapper FS Algorithm	Accuracy with Full Features	No. of Features Selected (out of 69)	Run Time (Sec)	Accuracy after Feature Selection	Iteration Number where the Best Solution Found
1	ACO-DT	99.144	7	22,339	99.053%	3
	CSO-DT		30	35.67	98.878%	6
	GWO-DT		22	20.499	99.008%	7
2	ACO-DT	99.189	9	22,901	99.17%	8
	CSO-DT		29	36.88	98.956%	8
	GWO-DT		24	19.31	98.988%	6
3	ACO-DT	99.163	10	24,458	99.163%	4
	CSO-DT		39	38.19	**99.202%**	4
	GWO-DT		23	19.127	99.079%	4
4	ACO-DT	99.131	12	28,960	**99.377%**	4
	CSO-DT		29	38.011	99.137%	8
	GWO-DT		23	19.501	**99.196%**	3
5	ACO-DT	99.286	10	25,066	99.202%	4
	CSO-DT		33	36.742	99.124%	5
	GWO-DT		24	19.176	99.170%	9
Avg.	ACO-DT	99.182	10	24,744.8	99.193%	4.6
	CSO-DT		32	37.098	99.059%	6.2
	GWO-DT		23.2	19.522	99.088%	5.8

Similar to the first experiment with the ClaMP dataset, we conducted five runs on all three bio-inspired wrapper feature selection algorithms, with DT as the classifier. For a fair comparison among the three, optimization algorithms all are run with 10 agents and 10 iterations and the free parameters as shown in Table 10. The average accuracy prior to feature reduction is 99.182%, the highest recorded being 99.286% in Run 5. It is worth noting that the accuracies before and after feature reduction are all par 99%, which proves the validity and fitness of our SOMLAP dataset when compared to the ClaMP dataset. Considering highest accuracy being the top priority, the best performance of the ACO-DT wrapper was obtained in Run 4, with 12 features out of 110 being selected, with a classifier performance of 99.377%. The best performance of CSO-DT was in Run 3, with 39 features and 99.202% accuracy. GWO-DT produced 23 features with 99.196% as its best post reduction accuracy in Run 4. As a proof of concept, the details of the features selected by each of the wrappers in Run 4 are listed in Table 15. To summarize the results of the evaluation of the SOMLAP dataset using three different swarm optimization algorithms, the average of five runs can be seen in Table 14. ACO-DT was the best, both in optimal feature selection and higher accuracy. However, the average ACO-DT run time was 24,744 s, which is 668 times higher than CSO-DT and 1267 times higher than GWO-DT. Clearly, ACO-DT proved to be more promising, with 10 out of 108 features selected on average and a 99.19% average accuracy.

Table 15. Details of features selected in Run 4 of Table 14.

FS Algorithm	No. of Features Selected. (With Best Accuracy)	Features Selected from DOS_Header (16 Features, 1–16)	Features Selected from COFF_Header (7 Features, 17–23)	Features Selected from Optional_Header (30 Features, 24–53)	Features Selected from Sections_Header (55 Features, 54–108)
ACO-DT	12	2, 10, 16	23	25, 28, 37, 44, 45, 46	80, 86
CSO-DT	29	1, 2, 6, 13, 16	22	24, 25, 28, 31, 35, 44, 45, 49, 50	55, 56, 67, 71, 73, 76, 85, 87, 89, 90, 91, 96, 97, 104
GWO-DT	23	11	18, 22	24, 25, 38, 40, 44	57, 58, 59, 61, 64, 66, 75, 76, 82, 84, 87, 89, 93, 102, 105.

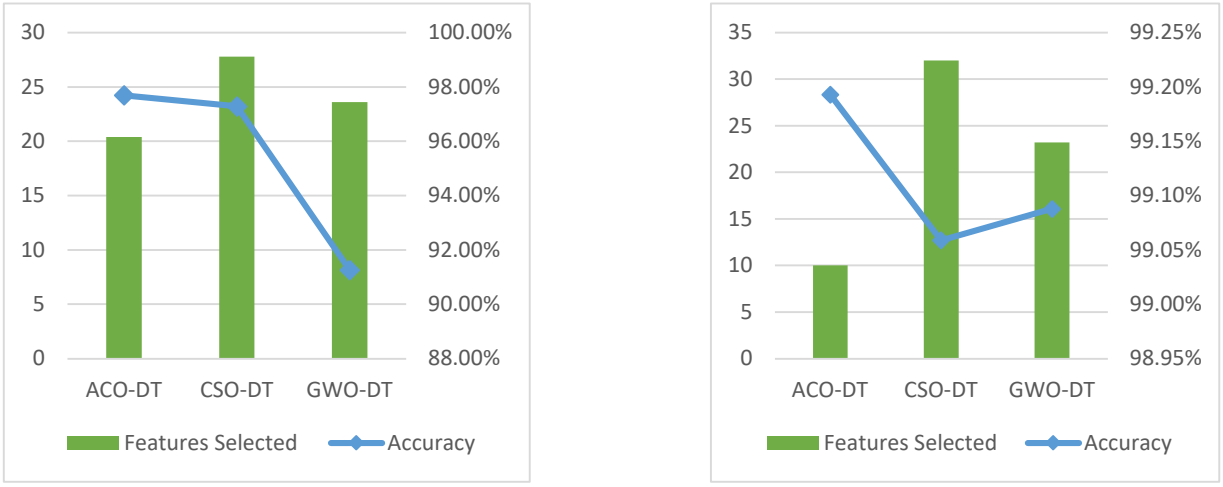
Comparison of results with similar works is tabulated in Table 16. Figure 4 portrays a graphical comparison of accuracies and features selected for both ClaMP and SOMLAP data sets.

Table 16. Comparison with similar works.

Sno	Paper	Number of Samples (units)	ML Technique	Feature Selection Technique	Number of Total Features	Number of Features after Reduction	Accuracy (After Reduction)
1.	Belaoued and Mazouzi [19] (2015)	552	NA	KHI square value	590	50	NA
2.	Salehi et al. [18] (2014)	1211	RF, J48 DT NB	Threshold Frequency	NA	6	98.4%
3.	Walenstein et al. [16] (2010)	23,906	NB, J48, SVM RF, IB5	Info. Gain	1867	15	99.8%
4.	Elovici et al. [15] (2007)	30,430	ANN, DT NB	Fischer Score	5500	300 5-g	95.5%
5.	kumar et al. [29] (2017)	5180	RF, LR, LDA DT, NB, kNN	NA	53 (Raw) + 68 (integ)	NA	NA
6.	Penmatsa et al. [30] (2020)	5180	RF, SVM, NB DT	ACORS algorithm	53 (Raw) + 68 (integ)	4 R + 2 I Features	90.55%
7.	Maleki et al. [27] (2019)	971	SVM, RF, NN, ID3, NB	forward feature selection	30	8	98.26%
8.	Vidyarthi et al. [23] (2017)	180	SVM	Info Gain	NA	232	92%
9.	Chen et al. [28] (2021)	1000 (malicious) + 1069 (benign)	XGBoost	PCA	NA	79	99.56%
10.	**This Paper**	**51,409 19,809 (malware) + 31,600 (benign)**	**RF, DT**	**ACO, CSO, GWO wrappers**	**108**	**12 (ACO-DT)**	**99.37%**

5.4. Discussion

The SOMLAP dataset was evaluated with several popular classifiers, and it was found that the DT classifier is the most optimized among them all. The average accuracy, in five runs, for the SOMLAP dataset was 99.18%, whereas for the ClaMP dataset it was 97.528%; this proves that the SOMLAP dataset proposed in this work has added value to the ClaMP dataset. In this work, another important contribution was to apply novel swarm optimization algorithms, Ant Colony Optimization (ACO), Cuckoo Search Optimization (CSO), and Grey Wolf Optimization (GWO), in wrapper mode with DT as the classifier, to find the most significant attributes in the SOMLAP dataset. The performance of all three algorithms was compared, and it was concluded that, as far as accuracy is considered, ACO-DT produced the highest, 99.37% after feature reduction. ACO-DT also outperformed others in dimensionality reduction, by selecting fewer attributes than others. As far as execution speed is considered, GWO-DT is the fastest algorithm, and almost near to ACO-DT in accuracy. The feature numbers identified by the optimization algorithms are revealed in Table 15 as empirical proof of the SOMLAP dataset.



(**a**) (**b**)

Figure 4. (**a**) Features selected vs. accuracy with the ClaMP dataset. (**b**) Features selected vs. Accuracy with the SOMLAP dataset.

From the experimentation and results obtained, the pros and cons of the three optimization algorithms are discussed. ACO, introduced in 1992, is an early swarm-based robust search technique. Here, the ant ensures that a local best solution is found as it traverses in the search space. ACO's main advantage is a guaranteed global optimum with fewer iterations and the possibility of parallel processing. The disadvantage is its high time complexity, due to incremental traversal from one node to another calculating the fitness each time. ACO has six tuning parameters, and a trial-and-error-based setting will consume more time. CSO, introduced in 2009, is a mimic of the brood parasitism of the cuckoo bird, and uses Levy flight for a possible solution selection. The advantage of CSO is that, since there is no node-to-node calculation of fitness, one iteration of CSO is very fast, and sometimes the stopping criteria may be reached early. However, CSO does not ensure the best result in fewer iterations. It has four tuning parameters to experiment with. GWO, introduced in 2014, is another random subset selection method that does not require a fitness calculation for the inclusion of each node to the solution and hence is faster per solution. However, similar to CSO, it does not ensure global optima for fewer iterations; it also needs more agents and more iterations compared to ACO. The advantage is its speed of convergence and only one tuning parameter.

6. Conclusions

PE file malware detection using ML tools, a hot topic of research, is taken up in this paper, with a goal of arriving at a new updated dataset over the current benchmark dataset. A critical survey of existing works in the area of malware identification based on ML methods was conducted and the scope for research was narrowed down. There is a need to construct a pure PE-header-field-based dataset that can contribute to improved detection rates. A new SOMLAP dataset (Swarm Optimization and Machine Learning Applied to PE Malware Detection) was developed, an extension over the ClaMP, benchmark dataset [12], to explore a possible improvement of malware detection accuracy. More samples, a total of 51,409, were collected. The malware sources were updated from Virus Share and benign sources from Windows 10 executables. We increased the attribute size to 108 compared to 69 in the ClaMP dataset by including several features from six sections of the section table. The dataset attributes are pure PE-header-based and are meant to prove the capability of PE header fields for efficient malware detection. The scope for improvement in the existing ClaMP dataset lies in the fact that the ClaMP dataset considered only the standard section of the PE header. In the SOMLAP dataset, we considered all the sections in the feature extraction. This fact is proven by the last column of Table 15, wherein the list of the

most significant features selected by the swarm optimization algorithms is shown from the various sections in the section table. The ClaMP dataset considered the entropy of only two sections, whereas, in our work, we considered the entropy of five sections. For the SOMLAP dataset, we excluded three features from the DoS header. Our hypothesis was that the common sections excluding text and data, such as bss, textbss, rsrc, and idata, have an impact on malware determination of a file. Our experimental results showed us that our hypothesis is right.

Swarm optimization algorithms were used to identify the most-contributing attributes of the SOMLAP dataset, aiming at reducing the features and improving the accuracy of detection. Three popular algorithms, viz., ACO, CSO, and GWO, with a decision tree classifier in wrapper mode, were compared with 10 agents and 10 iterations each for a fair comparison. The average number of features among five runs selected by ACO, CSO, and GWO were 10, 32, and 23 respectively. The average accuracies among five runs after feature reduction were 99.19%, 99.05%, and 99.08%, respectively. The average run times in seconds were 24,744, 37, and 19 respectively. Therefore, ACO outperformed others with respect to optimal feature reduction and accuracy. GWO outperformed with respect to runtime. All three algorithms were able to produce 99%+ accuracies.

As a future study, the SOMLAP dataset can be used by researchers to explore and compare other optimization algorithms and novel hybrid classifiers.

Author Contributions: Conceptualization, R.K.V.P.; methodology, R.K.V.P.; software, V.S.P.M. and S.J.K.; validation, S.J.K. and S.C.; formal analysis, S.C. and V.S.P.M.; investigation, S.J.K.; resources, R.K.V.P.; data curation, V.S.P.M. and S.J.K.; writing—original draft preparation, R.K.V.P.; writing—review and editing, S.J.K. and S.C.; visualization, S.C. and V.S.P.M.; supervision, R.K.V.P. and S.C.; project administration, R.K.V.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The new SOMLAP dataset developed in this research is available at https://www.kaggle.com/datasets/ravikiranvarmap/somlap-data-set (accessed on 23 December 2022). Researchers may download and use the dataset provided this paper is cited in their work.

Acknowledgments: The authors would like to thank the anonymous reviewers whose comments helped us to improve the quality of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Statcounter Global Stats—Browser, OS, Search Engine including Mobile Usage Share. Available online: https://gs.statcounter.com/os-market-share (accessed on 21 October 2022).
2. Damaševičius, R.; Venčkauskas, A.; Toldinas, J.; Grigaliunas, S. Ensemble-Based Classification Using Neural Networks and Machine Learning Models for Windows PE Malware Detection. *Electron* **2021**, *10*, 485. [CrossRef]
3. Pietrek, M. Peering Inside the PE—A Tour of the Win 32 Portable Executable File Format. *Microsoft Syst. J.* **1994**, *9*, 15–38.
4. Schultz, M.G.; Eskin, E.; Zadok, F.; Stolfo, S.J. Data Mining Methods for Detection of New Malicious Executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 14–16 May 2000.
5. Ye, Y.; Wang, D.; Li, T.; Ye, D.; Jiang, Q. An Intelligent PE-Malware Detection System Based on Association Mining. *J. Comput. Virol.* **2008**, *4*, 323–334. [CrossRef]
6. Choi, Y.-S.; Kim, I.-K.; Oh, J.-T.; Ryou, J.-C. PE File Header Analysis-Based Packed PE File Detection Technique (PHAD). In Proceedings of the International Symposium on Computer Science and its Applications, Hobart, TAS, Australia, 13–15 October 2008.
7. Wang, T.-Y.; Wu, C.-H.; Hsieh, C.-C. Detecting Unknown Malicious Executables Using Portable Executable Headers. In Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC, Seoul, Republic of Korea, 25–27 August 2009.
8. Wikibooks. PE Files. Available online: https://en.wikibooks.org/wiki/X86_Disassembly/Windows_Executable_Files#PE_Files (accessed on 21 October 2022).
9. Kim, S. *PE Header Analysis for Malware Detection*; San Jose State University Library: San Jose, CA, USA, 2019.

10. Namita; Prachi. PE File-Based Malware Detection Using Machine Learning. In *Proceedings of International Conference on Artificial Intelligence and Applications*; Springer: Singapore, 2021; pp. 113–123.
11. Wang, J.-H.; Deng, P.S.; Fan, Y.-S.; Jaw, L.-J.; Liu, Y.-C. Virus Detection Using Data Mining Techniques. In Proceedings of the IEEE 37th Annual 2003 International Carnahan Conference on Security Technology, Taipei, Taiwan, 14–16 October 2003.
12. Sung, A.H.; Xu, J.; Chavez, P.; Mukkamala, S. Static Analyzer of Vicious Executables (SAVE). In Proceedings of the 20th Annual Computer Security Applications Conference, Tucson, AZ, USA, 6–10 December 2004.
13. Kolter, J.Z.; Maloof, M.A. Learning to Detect Malicious Executables in the Wild. In *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining-KDD '04*; ACM Press: New York, NY, USA, 2004.
14. Moskovitch, R.; Stopel, D.; Feher, C.; Nissim, N.; Elovici, Y. Unknown Malcode Detection via Text Categorization and the Imbalance Problem. In Proceedings of the 2008 IEEE International Conference on Intelligence and Security Informatics, Taipei, Taiwan, 17–20 June 2008.
15. Elovici, Y.; Shabtai, A.; Moskovitch, R.; Tahan, G.; Glezer, C. Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 44–50.
16. Walenstein, A.; Hefner, D.J.; Wichers, J. Header Information in Malware Families and Impact on Automated Classifiers. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software, Nancy, France, 19–20 October 2010.
17. Ye, Y.; Li, T.; Jiang, Q.; Wang, Y. CIMDS: Adapting Postprocessing Techniques of Associative Classification for Malware Detection. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.* **2010**, *40*, 298–307.
18. Salehi, Z.; Sami, A.; Ghiasi, M. Using Feature Generation from API Calls for Malware Detection. *Comput. Fraud Secur.* **2014**, *2014*, 9–18. [CrossRef]
19. Belaoued, M.; Mazouzi, S. A Real-Time PE-Malware Detection System Based on CHI-Square Test and PE-File Features. In *IFIP Advances in Information and Communication Technology*; Springer International Publishing: Cham, Switzerland, 2015; pp. 416–425.
20. Akour, M.; Alsmadi, I.; Alazab, M. The Malware Detection Challenge of Accuracy. In Proceedings of the 2016 2nd International Conference on Open Source Software Computing (OSSCOM), Beirut, Lebanon, 1–3 December 2016.
21. Zatloukal, F.; Znoj, J. Malware Detection Based on Multiple PE Headers Identification and Optimization for Specific Types of Files. *J. Adv. Eng. Comput.* **2017**, *1*, 153. [CrossRef]
22. David, B.; Filiol, E.; Gallienne, K. Structural Analysis of Binary Executable Headers for Malware Detection Optimization. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 87–93. [CrossRef]
23. Vidyarthi, D.; Choudhary, S.P.; Rakshit, S.; Kumar, C.R.S. Malware Detection by Static Checking and Dynamic Analysis of Executables. *Int. J. Inf. Secur. Priv.* **2017**, *11*, 29–41. [CrossRef]
24. Raff, E.; Sylvester, J.; Nicholas, C. Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*; ACM: New York, NY, USA, 2017.
25. Sophos. Available online: https://www.sophos.com/de-de/medialibrary/PDFs/technical-papers/sophoslabs-machine-learning-tp.pdf (accessed on 21 October 2022).
26. Zhang, J. MLPdf: An Effective Machine Learning Based Approach for PDF Malware Detection. *arXiv* **2018**, arXiv:1808.06991.
27. Maleki, N.; Bateni, M.; Rastegari, H. An Improved Method for Packed Malware Detection Using PE Header and Section Table Information. *Int. J. Comput. Netw. Inf. Secur.* **2019**, *11*, 9–17. [CrossRef]
28. Chen, Z.; Zhang, X.; Kim, S. A Learning-Based Static Malware Detection System with Integrated Feature. *Intell. Autom. Soft Comput.* **2021**, *27*, 891–908. [CrossRef]
29. Kumar, A.; Kuppusamy, K.S.; Aghila, G. A Learning Model to Detect Maliciousness of Portable Executable Using Integrated Feature Set. *J. King Saud Univ. Comput. Inf. Sci.* **2019**, *31*, 252–265. [CrossRef]
30. Penmatsa, R.K.V.; Kalidindi, A.; Mallidi, S.K.R. Feature Reduction and Optimization of Malware Detection System Using Ant Colony Optimization and Rough Sets. *Int. J. Inf. Secur. Priv.* **2020**, *14*, 95–114. [CrossRef]
31. Virusshare. Available online: https://virusshare.com/ (accessed on 21 October 2022).
32. Pefile. PyPI. Available online: https://pypi.org/project/pefile/ (accessed on 21 October 2022).
33. Microsoft. Karl-Bridge-Microsoft. PE Format. Available online: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format (accessed on 21 October 2022).
34. Wikibooks. x86 Disassembly/Windows Executable Files. Available online: https://en.wikibooks.org/wiki/X86_Disassembly/Windows_Executable_Files (accessed on 21 October 2022).
35. Chen, R.-C.; Dewi, C.; Huang, S.-W.; Caraka, R.E. Selecting Critical Features for Data Classification Based on Machine Learning Methods. *J. Big Data* **2020**, *7*, 1–26. [CrossRef]
36. Keogh, E.; Mueen, A. Curse of Dimensionality. In *Encyclopedia of Machine Learning and Data Mining*; Springer: Boston, MA, USA, 2017; pp. 314–315.
37. Tabakhi, S.; Moradi, P.; Akhlaghian, F. An Unsupervised Feature Selection Algorithm Based on Ant Colony Optimization. *Eng. Appl. Artif. Intell.* **2014**, *32*, 112–123. [CrossRef]
38. Vanaja, R.; Mukherjee, S. Novel Wrapper-Based Feature Selection for Efficient Clinical Decision Support System. In *Advances in Data Science*; Springer: Singapore, 2019; pp. 113–129.
39. Dorigo, M.; Di Caro, G. Ant Colony Optimization: A New Meta-Heuristic. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999.

40. Gambardella, L.M.; Dorigo, M. Solving Symmetric and Asymmetric TSPs by Ant Colonies. In Proceedings of the IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 20–22 May 1996.
41. Blum, C.; Sampels, M. Ant Colony Optimization for FOP Shop Scheduling: A Case Study on Different Pheromone Representations. In Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600), Honolulu, HI, USA, 12–17 May 2002.
42. Al-Ani, A. Ant Colony Optimization for Feature Subset Selection. *Int. J. Comput. Inf. Eng.* **2007**, *1*, 999–1002.
43. Sivagaminathan, R.K.; Ramakrishnan, S. A Hybrid Approach for Feature Subset Selection Using Neural Networks and Ant Colony Optimization. *Expert Syst. Appl.* **2007**, *33*, 49–60. [CrossRef]
44. Yang, X.-S.; Deb, S. Engineering Optimization by Cuckoo Search. *arXiv* **2010**, arXiv: 1005.2908.
45. Aziz, M.A.E.; Hassanien, A.E. Modified Cuckoo Search Algorithm with Rough Sets for Feature Selection. *Neural Comput. Appl.* **2018**, *29*, 925–934. [CrossRef]
46. Alia, A.F.; Taweel, A. Feature Selection Based on Hybrid Binary Cuckoo Search and Rough Set Theory in Classification for Nominal Datasets. *Int. J. Inf. Technol. Comput. Sci.* **2017**, *9*, 63–72. [CrossRef]
47. Wang, G. A Comparative Study of Cuckoo Algorithm and Ant Colony Algorithm in Optimal Path Problems. *MATEC Web Conf.* **2018**, *232*, 03003. [CrossRef]
48. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey Wolf Optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
49. Emary, E.; Zawbaa, H.M.; Hassanien, A.E. Binary Grey Wolf Optimization Approaches for Feature Selection. *Neurocomputing* **2016**, *172*, 371–381. [CrossRef]
50. Al-Tashi, Q.; Abdul Kadir, S.J.; Rais, H.M.; Mirjalili, S.; Alhussian, H. Binary Optimization Using Hybrid Grey Wolf Optimization for Feature Selection. *IEEE Access* **2019**, *7*, 39496–39508. [CrossRef]