Malware Detection System using Machine Learning and DATA-Mining Techniques

P.Sujatha, S.Sivasankari, P.SriPriya, R.Devi, K.Sharmila

Abstract:A serious threat these days is malicious executables. It's designed to wreck computing system and a few of them cover network while not the information of the owner victimisation the system. Two approaches are derived for it i.e. Signature primarily based Detection and Heuristic primarily based Detection. These approaches performed well against celebrated malicious programs however cannot catch the new malicious programs. Totally different researchers have planned ways victimisation data processing and machine learning for police investigation new malicious programs. The strategy supported data processing and machine learning has shown sensible results compared to alternative approaches. This work presents static malware detection system victimisation data processing techniques like data Gain, Principal part analysis, and 3 classifiers: SVM, J48, and Naïve mathematician. For overcoming the dearth of usual anti-virus product, this paper has a tendency to use ways of static analysis to extract valuable options of Windows letter file as well as to extract raw options of Windows executables that area unit letter header data, DLLs, and API functions within every DLL of Windows letter file. Thereafter, data Gain, job frequencies of the raw options area unit calculated to pick out valuable set options, so principal part analysis is employed for spatial property reduction of the chosen options. By adopting the ideas of machine learning and data-mining, this research work constructs a static malware detection system that features a detection rate of 99.6%.

Keywords: ***Malware Detection, Malicious Codes, Malware, Malware Detection, data Security, data processing***

I. INTRODUCTION

Cohen 1st formalized the term bug in 1983; it's evolved into malware detection. Malicious programs, ordinarily termed as malware, will be classified into Virus, Trojans, Backdoors, and malevolent program and a spread of alternative categories. Laptop malware remains a serious threat to today's laptop systems and networks. In line with a recent threat report by Symantec, trends for 2011, created 403 million new malicious code signatures, that could be a huge increase over 2010, once 286 new malicious code signatures were side.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
Dr.P.Sujatha*, Professor, School of Computing Sciences, VISTAS
S.Sivasankari, Research Scholar, School of Computing Sciences, VISTAS
Dr.P.SriPriya, Professor, School of Computing Sciences, VISTAS
R.Devi, Associate Professor, School of Computing Sciences, VISTAS
Dr.K.Sharmila, Associate Professor, School of Computing Sciences, VISTAS

Although commonest detection technique may be a signature primarily based detection that creates the core of most business anti-virus programs, it's not effective against "zero-day attacks" or antecedently unknown malwares [1].
The main target of malware researchers has shifted to seek out a lot of generalized and scalable options which will establish antecedently unknown malwares rather than a static signature.

The [2] two major approaches for malware analysis are static and dynamic malware analysis. Within the static analysis, code and structure of a program is examined while not truly running the program whereas dynamic analysis the program is dead during a real or virtual atmosphere. This research work proposes static malware detection system victimization data processing techniques that it mechanically extracts options of malware and cleans programs and uses them to classify programs into malicious or benign executables. Main objective within the analysis of this method is to simulate the task of detective work new malicious executables.

II. RELATEDi WORKS

This paper [5] has considered the details of most relevant malware detection techniques in this section. In recent years several malware researchers have cantered on data processing to notice unknown malwares. Methoding is that the process of analyzing electronically keeps knowledge by mechanically checking out patterns. Machine learning algorithms are used wide for various data processing issues to notice patterns and to seek out correlations between knowledge instances and attributes. Several researchers have used n-grams or API calls as their primary variety of feature that square measure accustomed represent malware instances during an appropriate format for data processing functions.

Shultz et al. [6] projected way victimization data processing techniques for detective work new malicious executables. 3 differing types of options square measure extracted from the executables, i.e. the list of DLLs utilized by the binary, the list of DLL operate calls, and variety of various system calls used inside every DLL. Conjointly they associate degreealyze computer memory unit sequences extracted from the hex dump of a workable. The info set consisted of four, 266 files out of that three, 265 were malicious and one, 001 were legitimate or benign programs. A rule induction algorithmic program referred to as murderer [7] was applied to seek out patterns within the DLL knowledge.

A learning algorithmic program Naïve mathematician (NB) that relies on theorem statistics was accustomed notice patterns within the string knowledge and n-grams of computer memory unit sequences were used as input file for the Multinomial Naïve mathematician algorithmic program. A knowledge set is partitioned off in 2 data sets, i.e., a check knowledge set and a coaching knowledge set. This is often to permit for performance testing on knowledge that square measure freelance from the info accustomed generates the classifiers.

The Naïve mathematician [16] formula, mistreatment strings as input file, yielded the very best classification performance with associate degree accuracy of 95%. The authors compared their results with ancient signature-based ways and claimed that the information mining-based detection rate of recent malware was doubly as high as compared to the signature-based formula.

A similar approach was employed by J. Z. Kolter et al. [8], wherever they use n-gram analysis and data processing approaches to find malicious executables within the wild. The authors used a hex dump utility to convert every workable to positional notation code in associate degree code format and created n-gram options by combining every four-byte sequence into one term. Their primary dataset consisted of 1971 clean and 1651 malicious programs They used completely different classifiers as well as Instance-primarily based Learner, TFIDF, Naive-Bayes, Support vector machines, call tree, boosted Naive-mathematician, SVMs and boosted call tree. They used info gain to pick valued options that ar provided as input to any or all classifiers. The realm below associate degree mythical monster curve (AUC) may be a lot of complete live compared with the detection accuracy as they rumored [9]. AUCs show that the boosted call trees shell remainder of the classifiers for each classification issues.

M. Siddiqui et al. [10] used data processing for detection of Worms. They used variable length instruction sequence. Their Primary knowledge set consists of two, 775 Windows alphabetic character files, within which within which 1400 were worms and one, 330 were benign. They performed detection of compilers, common packers and crypto before dismantling of files. Sequence reduction was performed and ninety seven of the sequences were removed. They used call Tree, textile and Random Forest models mistreatment. Random forest performed slightly higher than the others.

III. MALWAREi DETECTIONi SYSTEM'Si ARCHITECTURE

This section describes the architecture of the proposed system. This technique is to accurately observe new malware (unknown malware) binaries employing a variety of information mining techniques. Fig. 1 shows the design of the malware detection system. The system consists of 3 main modules: PE-Miner, Feature selection and information transformation, and Learning algorithms.

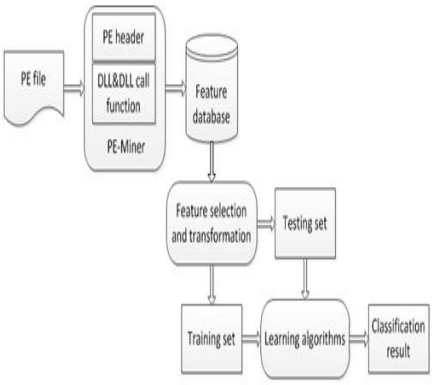

Fig. 1. Architecture of Malware Detection System

- **Module 1:** PE-Miner parses PE tables of all executables to find out all PE header information, DLL names, and API functions inside each DLL as raw features;
- **Module 2:** Information Gain value of each PE header, and calling frequency of each DLL and API function inside DLL are computed and those raw features with Information Gain values and calling frequencies greater than a threshold are selected; then Principal Component Analysis is applied to further transform and reduce the number of features;
- **Module 3:** According to the features after PCA, transform every program in the database to its corresponding feature vector and then use a learning algorithm to derive a classification result from these labelled feature vectors;

A. Dataset Description

The dataset consists of 247348 Windows programs in letter file format those square measure 236756 malicious and 10592 clean programs. There have been no duplicate programs in the dataset. The malicious programs were obtained from many sources like VX Heavens Virus assortment information that is offered without charge transfer within the property right. The numbers of malicious programs square measure sorted by their corresponding name in Table I . The clean programs were collected from transfer.com [11], Windows' system files and program files.

This research work solely contemplates non-packed Windows binary files in letter format. Some malware writers compress their code to form them undetectable by antivirus programs. Therefore this research tends to use variety of most well-liked tools to decompress the files, that square measure compressed, like UPX, ASPack, PECompact, etc. that build the letter files recognizable by letter programme. Also this work tends to develop a letter file programme named letter jack that extracts all letter format file's header info, DLLs, and API functions within every DLL of the programs within the dataset.

Table – I: Variety of malicious sorted by their cognomen

Name of malicious program	Counts
Trojan-IM	481
Packed	379
IM-Worm	379
not-virus_Hoax	318
Trojan-Mailfinder	301
Flooder	267
DoS	257
Email-Flooder	196
Trojan-DDoS	135
IM-Flooder	134
Trojan-Ransom	94
SpamTool	64
Trojan-Notifier	59
SMS-Flooder	57
Spoofer	26
not-virus_BadJoke	20
Trojan-SMS	14
Unknown	2
Sniffer	2
not-a-virus_NetTool	1
not-a-virus_RemoteAdmin	1

B. Feature Extraction

The information of degree practicable is also merely derived its letter file format whereas not running programs [12, 13]. This work develops degree application named "PE-Miner" to analyze letter tables of Windows practicable programs among the collected dataset. The letter jack extracts all letter header information, DLL names, and API operates names referred to as within those DLLs contained throughout a letter file. The PE-Miner program stores all the extracted information throughout a piece of information. Throughout this study, the PE-Miner program extracted all raw choices of 236756 malicious and 10592 clean programs and keeps them throughout a piece of information. Different unit of measurement is summarized in Table II.

Table – II: Raw Features

¼	Description	Data Type	Quantity
1	Unique DLL features	Binary	792
2	Unique API function calls	Binary	24,662
3	PE DOS header	Integer	31
4	PE file header	Integer	7
5	PE optional headers	Integer	30
6	PE data directories	Integer	16
7	PE section headers	Integer	10
8	PE import description	Integer	5
9	PE export description	Integer	11
10	PE resource table	Integer	6
11	PE debug info	Integer	8
12	PE delay import	Integer	8
13	PE TLS table	Integer	6
Total			**25,592**

Fig. 2 shows the main graphics program (GUI) of the letter-Miner that extracts raw choices from letter files throughout alike folder and automatically stores all choices throughout a piece of information.

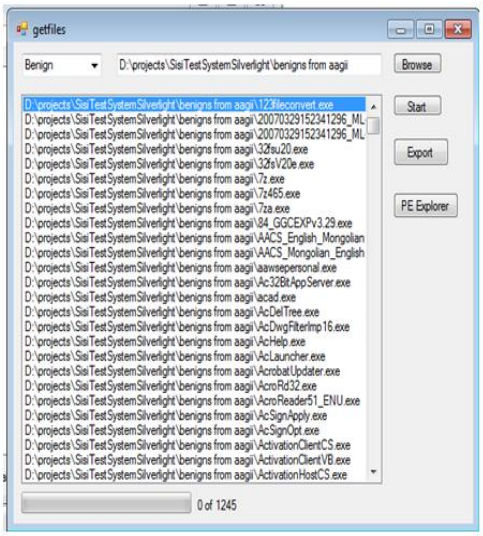

Fig. 2. Main GUI

C. Feature Alternative

The data ought to be pre-processed to select out a group of the attributes to use in learning. By following learning ways, performance can be improved by preselecting, however learning ways that themselves attempt to select attributes and ignore extraneous or redundant ones [14].

After raw choices of all practicable programs at intervals the information area unit extracted practice PE-Miner program, the system computes each letter header feature's information Gain price and business frequency of DLLs and API functions. The top-20 letter header choices class-conscious by antibody on dataset unit of measurement bestowed in Table 3.

Table – III: Top Features of PE Header

¼	Features	Mean values	
		Malware	Benign
1	SizeOfInitializedData	59,675,152	600,081
2	NumberOfSymbols	59,251,515	1,655,386
3	certificateTableSize	103,615,710	388,027
4	exporttableSize	8,350,672	1,702
5	IATSIZE	2,341,313	509
6	BoundImportSize	68,707,156	200,506
7	LoadConfigTableSize	86,574,977	306,799
8	BaseRelocationTableSize	58,377,700	228,904
9	e_cp	1,273	5
10	CLRRuntimeHeaderSize	49,875,208	135,841
11	VirtualSize	60,309,693	301,690
12	SizeOfRawData	80,791,328	1,817,641
13	NumberOfLineNumbers	1,675	3
14	e_cblp	1,353	143
15	e_minalloc	1,118	2
16	NumberOfRelocations	1,167	67
17	AddressOfFuncitons	2,884,543	378,963
18	AddressOfNameOrdinals	2,538,096	382,150
19	AddressOfNames	2,624,183	11,265
20	E_ovno	1,409	10

As shown in Table III, there's a big distinction between the values of major linker versions and also the size of the initialized knowledge within the benign and malicious executables.

The scale of the initialized knowledge in benign executables is typically considerably higher compared to those of the malicious executables. It's fascinating to notice that an affordable variety of symbols area unit gift in benign executables. It will be seen in Table 3 that the values of fields like variety of relocation and variety of line numbers area unit considerably higher within the malware workable as compared. Note that the amount of functions is comparatively higher for the malware executables.

Considering each DLL as a feature appears to be helpful as a result of a program's DLLs will tell you a great deal regarding its practicality. The top-30 DLL name options hierarchic by business frequency on dataset area unit conferred in Table IV. Most ordinarily used DLL is Kernel32.dll that contains core practicality, like access and manipulation of memory, files, and hardware.

Table- IV: Frequency of DLL

No	DLLs	Frequency	Description
1.	kernel32.dll	3,467,435	Windows NT BASE API Client DLL
2.	user32.dll	912,115	USER API Client DLL
3.	advapi32.dll	441,914	Advanced Win32 application programming interfaces
4.	gdi32.dll	201,687	GDI Client DLL, involved in graphical displays.
5.	wininet.dll	84,100	Internet Extensions for Win32
6.	comctl32.dll	72,010	User Experience Controls Library
7.	shell32.dll	55,169	Windows Shell Common DLL
8.	wsock32.dll	49,133	Windows Socket 32-Bit DLL
9.	oleaut32.dll	49,060	OLE 2 - 32 automation
10.	msvbvm50.dll	44,851	Visual Basic Virtual Machine
11.	ole32.dll	43,531	32-bit OLE 2.0 component
12.	shlwapi.dll	38,718	Library for UNC and URL Paths, Registry Entries
13.	ws2_32.dll	22,486	Windows Socket 2.0 32-Bit DLL
14.	ntdll.dll	18,570	Win32 NTDLL core component
15.	urlmon.dll	16,117	OLE32 Extensions for Win32
16.	version.dll	12,481	Version Checking and File Installation Libraries
17.	crtdll.dll	11,420	Microsoft C Runtime Library
18.	comdlg32.dll	10,484	Common Dialogue Library
19.	winmm.dll	8,793	Windows Multimedia API
20.	rpcrt4.dll	5,531	Remote Procedure Call Runtime
21.	psapi.dll	5,228	The process status application programming interface
22.	msvcr100.dll	4,702	Microsoft Visual C++ Redistributable DLL
23.	hal.dll	3,993	The Windows Hardware Abstraction Layer
24.	mpr.dll	3,670	Multiple Provider Router DLL
25.	netapi32.dll	3,510	Net Win32 API DLL
26.	avicap32.dll	3,177	AVI capture API
27.	rasapi32.dll	2,973	Remote Access 16-bit API Library
28.	cygwin1.dll	2,803	Cygwin® POSIX Emulation DLL
29.	mscoree.dll	2,774	Microsoft .NET Runtime Execution Engine
30.	imagehlp.dll	2,049	Windows NT Image Helper

Using Windows' advanced core parts like the Service Manager and written record area unit provided by Advapi32.dll, User32.dll contains all the user-interface parts, like buttons, scroll bars, and parts for dominant and responding to user actions. Gdi32.dll contains functions for displaying and manipulating graphics. Executables usually don't import Ntdll.dll directly, though it's continuously foreign indirectly by Kernel32.dll. It's a file containing the interface to the Windows kernel. If associate workable imports this file, it means the author supposed to use practicality not commonly offered to Windows programs. Some tasks, like concealment practicality or manipulating processes, can use this interface. WSock32.dll and Ws2_32.dll area unit networking DLLs. Programs connect with a network or perform network-related tasks that access either of those files. Higher-level networking functions that implement protocols like FTP, HTTP, and NTP area unit contained in Wininet.dll. The last domain of relevant options is API decision classes.

As a result of the experimental analysis on the malicious executables in the dataset, this work identifies seven main groups of commonly used API function call features that are based on the malicious behaviours and these are listed below:

- **File-related behaviour:** Create a specific file in sensitive folders; Delete, break, cover system files or application files; Edit file; Traverse file directory and search for target files;
- **Process-related behaviour:** Release DLL file to inject into system process and create a new thread to hide itself; Create new processes to execute its code; Create threads to search and terminate the process of anti-virus or protection software; Create a matrix process to prevent repeated execution;
- **Memory-related behaviour:** Free, move and replace spaces of the memory; Occupy extra memory space, decrease the total available memory size; Forbid memory allocation and reclaim memory space; Point the interrupt vector address to the initial address of the malicious code;
- **Register-related behaviour:** Add or delete system service; Auto run while the system is starting up; Hide and protect itself; Undermine system function;
- **Network-related behaviour:** Open or listen specific port; Send through chatting software; Steal information and sending them to the mailbox; Enumerate weak password vulnerable computers; Occupying resources;
- **Windows service behaviour:** Terminating windows update service; terminating windows firewall; Opening Telnet service;
- **Other behaviours:** Hooking keyboard; Hiding window; Alter system time to disable software prevention; Restart the computer; Scan existing vulnerabilities of the system;

To interpret these features, when going back to malware definitions and their known major behaviours we can conclude that file management is the most important activity in viruses, because this kind of malware is known to copy itself multiple times and mutate itself every now and then if it's a polymorphic/metamorphic malware. Another difference when studying malware and benign applications is the type of process management done by malicious code that is intended to hide from, or spoof antivirus tools and process monitors. Since getting hidden is more complicated than running normally, this approach needs more calls to process management procedures. The third discriminative category is the Memory, which shows the difference that malicious executable intends to free, move and replace spaces of the memory.

Next category is the Registry, malicious executables have heavily used the Windows Registry to change system menus, and user privileges, start-up programs, and file extension handling attributes to ease malware distribution or to destruct the target computer and so on next category behaviours are activated.

According to PE header, DLL, and API function's feature sets constructed above, feature vector could be used to represent each program Pi. Each vector element corresponds to a feature Fk. And its value indicates whether that executable has the corresponding feature or not. With these values, the feature vector F for a program P is defined as follows:

$$F = \{F_1, F_2, F_3, F_N\}$$

IV. IMPLEMENTATIONi OFi PROPOSEDi METHOD

This work implements a programme named misdetection that is integrated many approaches that was mentioned in the previous sections. Therefore, first, this work exports a model of trained J48 classifier from the rail. So the model is integrated and therefore different functions that extract, select, and scale back options of letter of the alphabet file, into a programme. Fig. 3 shows main user interface of the misdetection program.

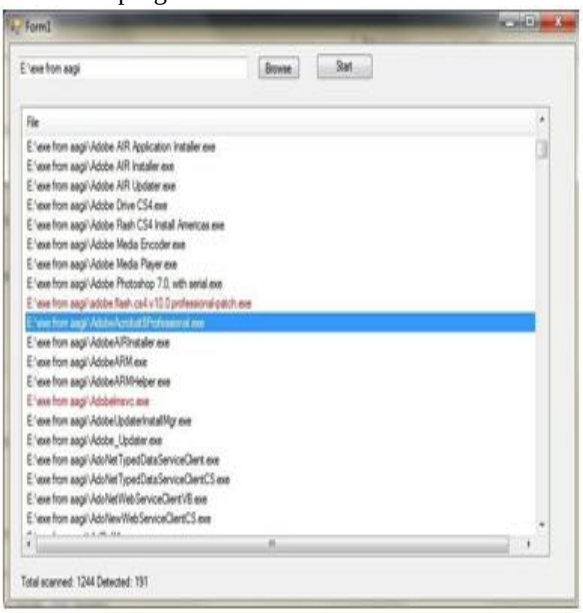

Fig.3. Main GUI for Malware detection

The misdetection program's operating steps are illustrated below,

Step 1: It browses such as directory and its subdirectories to search out files within the directory; if 'a' go into the directory isn't letter of the alphabet format file, it simply ignores the file and continues its work;

Step 2: It parses letter of the alphabet tables of the viable to search out all letter of the alphabet header data, DLL names, and API functions within every DLL as raw features;

Step 3: It selects the required options from the raw features;

Step 4: Principal part Analysis is applied to any rework and scale back the amount of features;

Step 5: It transforms a program's information to its corresponding feature vector; so apply to a classification algorithm;

Step 6: It prints the classification results as shown at all-time low of Fig. 4;

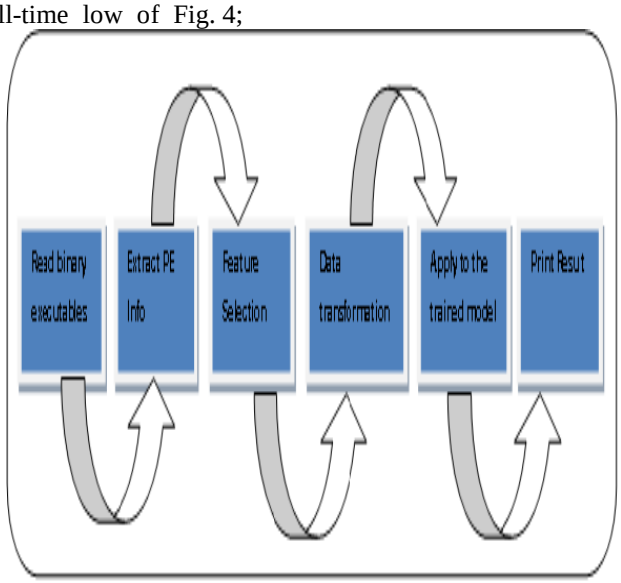

Fig.4. Working Model

V. EXPERIMENTALi RESULTS

The numbers of malicious programs are grouped by their corresponding family name in Table 1. The clean programs were collected from Download.com, freshly installed Windows's system files and local laboratories. This is a comprehensive database that contains many thousands of malware samples. The samples consist of Virus, Worm, Trojan, Backdoors, and Spyware etc.,

This work only considers non-packed Windows binary files in PE format. Some malware writers compress their code to make them undetectable by antivirus programs. So this work first used a number of the most popular tools to decompress the files, which are compressed, such as UPX, ASPack, PECompact, etc. that make the PEs recognizable by the PE parser. To evaluate the performance, the following classifiers are selected: SVM, J48, and NB rule learner. All of these classifiers are available as part of the Java-based open source machine learning toolkit Weka [14]. In this research, the experiments were done on an Intel Pentium Core 2 Duo 2.33 GHz processor with 2GB RAM. The Microsoft Windows 7 SP1 is installed on this machine. The classifiers (SVM, J48, NB) are trained and tested with cross-validation 10 folds, i.e., the dataset is randomly divided into 10 smaller subsets, where 9 subsets are used for training and 1 subset is used for testing. The process is repeated 10 times for every combination. This methodology helps in evaluating the robustness of a given approach to detect malicious Win32 files that contain malware without any a priori information.

There were total 138 PE header, 792 DLL, and 24662 API function's raw features in the feature set of this work.

Some experiment is done to select a subset of features by the detection rate of the system. Number of features with different ranges was selected; system is trained and tested it to see the overall accuracy. According to the result of the analysis, the top 88 PE header, 130 DLL, and 2453 API function features were finally selected to train our system which had better performances with the classifiers.

A. Performance Evaluation Criteria

The standard 10-fold cross-validation process is used in this experiment: the dataset is randomly divided into 10 smaller subsets, where 9 subsets are used for training and 1 subset is used for testing. The process is repeated 10 times for every combination. This work focused in the detection rate of the classifiers and this was the percentage of the total malicious programs labelled malicious.

True Positives (TP): the number of malicious executable examples classified as malicious executable

True Negatives (TN): the number of benign programs classified as benign

False Positives (FP): the number of benign programs classified as malicious executable

False Negatives (FN): the number of malicious executable classified as benign executable

We were interested in the detection rate of the classifiers. In our case this was the percentage of the total malicious programs labelled malicious. We were also interested in the false positive rate. This was the percentage of benign programs which were labelled as malicious, also called false alarms.

True positive rate is also called the Detection Rate (DR) and defined as,

$$DR = \frac{TP}{TP+FN} * 100\%$$

False Positive Rate (FPR) as,

$$FPR = \frac{FP}{TN+FP} * 100\%$$

Overall Accuracy (OA) as,

$$OA = \frac{TP+TN}{TP+TN+FP+FN} * 100\%$$

Evaluating the detection efficiency of the system, it is not only considering the higher TP rate but also considering the lower FP rate. Therefore, the overall accuracy could be the main reference for us to compromise both TP rate and FP rate.

There were total 138 PE header, 792 DLL, and 24662 API function's raw features in our feature set. We did some experiment to select a subset of features by the detection rate of the system. We just selected a number of features with different ranges, trained the system, and tested it to see the overall accuracy.

According to the result of our analysis, we finally selected the top 88 PE header, 130 DLL, and 2453 API function features to train our system which had better performances with the classifiers. A number of selected subset features of the experiments are given in Table 3. A list of the full classification results are shown in Table V and Table VI.

Table – V: System performance based on individual Features

Feature type	Classifier	OA (%)
PE HEADER	Naïve-Bayes	77
	SVM	80
	J48	**99**
API FUNCTIONS	Naïve-Bayes	97
	SVM	81
	J48	**99.1**
DLL	Naïve-Bayes	89
	SVM	91
	J48	**96.9**

System performance based on individual features is represented through bar chart as given below.

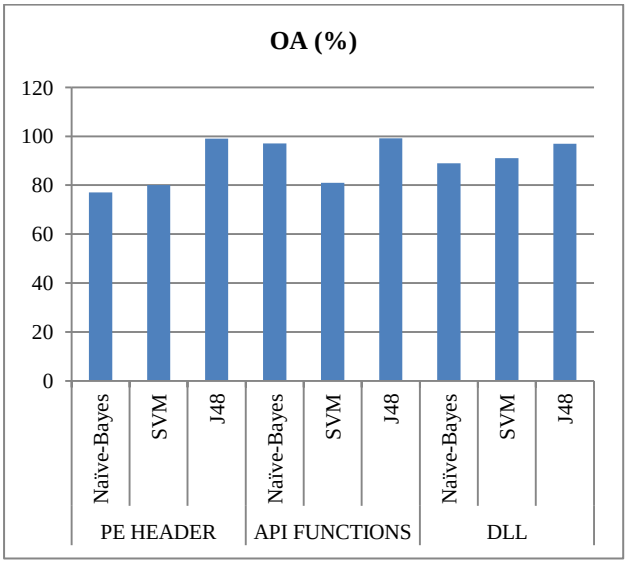

Fig.5. System performance based on individual features

Table-VI: System performance based on Hybrid Features

Feature type	Classifier	OA (%)
PE HEADER&DLL	Naïve-Bayes	72
	SVM	85
	J48	**90**
PE HEADER & API FUNCTIONS	Naïve-Bayes	93
	SVM	97
	J48	**99**
PE HEADER, DLL & API FUNCTIONS	Naïve-Bayes	93
	SVM	97
	J48	**98**

System performance based on different type of combined features is represented through bar chart as given below.

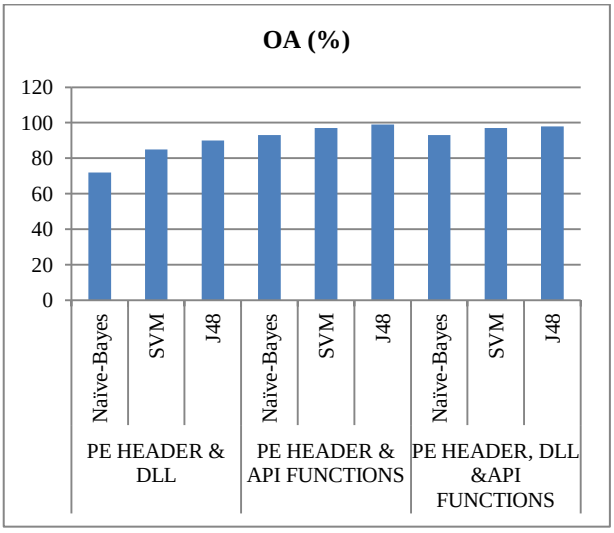

Fig.6. System performance based on combined features

As shown in Table V and Table VI, the detection accuracy of J48 classifier outperforms the other classifiers in all the cases. It is concluded from the above table that the classifiers victimization the DLL name feature had the bottom detection rate. Further, Naïve-Bayes provides the worst detection accuracy in most cases. The classifiers victimization combined letter of the alphabet header and API perform options provide slightly higher detection accuracy than the classifiers victimization letter of the alphabet header feature alone. Because of this result it is concluded that the letter of the alphabet header feature while not combining with the opposite options is effectively used for police work zero day malwares. Moreover, a classifier victimization letter of the alphabet header feature will have the tiniest process overheads.

VI. CONCLUSION

This work bestowed static malware detection system victimization data processing techniques. this technique is predicated on mining letter of the alphabet header data, DLL name, API perform calls within every DLL options of the executables for police work malware and malicious codes. This work tends to develop a letter of the alphabet- labourer program to dissect PE format of the Windows executables in our dataset. The letter of the alphabet labourer extracts all letter of the alphabet header data, DLL names, and API perform calls within every DLL contained in an exceedingly letter of the alphabet file. And this program stores all the extracted data in an exceedingly information.

After raw options of all viable programs within the information are extracted victimization PE-Miner program, the system computes every letter of the alphabet header feature's data Gain values and counts the career frequencies of DLLs and API functions to pick set feature sets. Consistent with the results of the analysis, this work finally takes high eighty eight letter of the alphabet header, 130 DLL, and 2453 API perform options to train the system that had higher performances with the classifiers.

In order to grasp the contribution of PCA transformation, PCA perform was removed from the system, i.e., the chosen options were directly applied to coach the classifiers. After PCA perform is enabled, we have a tendency to go higher results and therefore the coaching time was abundant lessened from some of days to many hours thanks to the amount of options being reworked and reduced, and therefore the performances were raised further. Finally, three different sorts of options are mixed into one feature set which is used to decide the hybrid feature set (HFS), and these options are used to train SVM, J48, and NB classifiers severally for detection of Windows letter of the alphabet malwares. As shown in the experimental results, the detection accuracy of J48 classifier outperforms the remainder of the mining classifiers in most of the cases. And, the classifiers victimization the DLL name feature had the bottom detection rate. Further, Naïve-Bayes provides the worst detection accuracy in most cases. The classifiers victimization combined letter of the alphabet header and API perform options provide slightly higher detection accuracy than the classifiers victimization letter of the alphabet header feature alone. It is concluded that the letter of the alphabet header feature while not combining with the opposite options is effectively used for police work zero day malwares. Moreover, a classifier victimization letter of the alphabet header feature will have the tiniest process overheads. The experimental result shows that the proposed system provides most effective detection accuracy of 99.6% with the false positive rate 2.8%.

REFERENCES

1. Ravi, C & Manoharan, R. Malware Detection using Windows Api Sequence and Machine Learning. International Journal of Computer Application, Vol.43, No.17, 2012.
2. Ravi, C & Chetia, G. Malware Threats And Mitigation Strategies: A Survey, Journal of Theoretical and Applied Information Technology, Vol. 29, No. 2, pp. 69-73, 2011.
3. Egele, M. S, A Survey on Automated Dynamic Malware-Analysis. ACM Computing Surveys, Vol. 44, No. 2, 2012.
4. Herath, H. M. P. S., & Wijayanayake, W. M. J. I. Computer Misuse in the Workplace. Journal of Business Continuity & Emergency Planning, Vol.3, No.3, P.P 259–270, 2009.
5. I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed. Morgan Kaufmann, 2005.
6. M. G. Schultz, E. Eskin, E. Z., and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in Proceedings of the IEEE Symp. on Security and Privacy, pp. 38-49, 2001.
7. W. Cohen, ."Fast effective rule induction,." Proc. 12th International Conference on Machine Learning, pp. 115-23, San Francisco, CA: Morgan Kaufmann Publishers, 1995.
8. J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables in the wild," in Proceedings of the ACM Symp. on Knowledge Discovery and Data Mining (KDD), pp. 470-478, August 2004.
9. T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Researchers", TR HPL-2003-4, HP Labs, USA, 2004.
10. M. Siddiqui, M. C. Wang, J. Lee, Detecting Internet worms Using Data Mining Techniques, Journal of Systemics, Cybernetics and Informatics, volume 6 - number 6, pp: 48-53, 2009.

11. Mathur, K., and Saroj H. A Survey on Techniques in Detection and Analyzing Malware Executables. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 44, No. 2, 2012.
12. Doherty, N. F., Anastasakis, L., & Fulford, H, The Information Security Policy Unpacked: A Critical Study of the Content of University Policies. International Journal of Information Management, Vol.29, No.6, pp. 449–457, 2009.
13. G. Tahan, L.R.Y. Automatic Malware Detection Using Common Segment Analysis and Meta-Features. Journal of Machine Learning Research, 13l, pp. 949-979, 2012.
14. I. Gurrutxaga , Evaluation of Malware clustering based on its dynamic behaviour. Seventh Australasian Data Mining conference, Australia, pp. 163–170, 2008.
15. Patel, S. C., Graham, J. H., & Ralston, P. A, Qualitatively Assessing the Vulnerability of Critical Information Systems: A New Method for Evaluating Security Eenhancements. International Journal of Information Management, Vol.28, pp. 483–491, 2008.
16. Osama Mohammed Qasim , Karim Hashim Al-Saedi, "Malware Detection using Data Mining Naïve Bayesian Classification Technique with Worm Dataset" in International Journal of Advanced Research in Computer and Communication Engineering - Vol. 6, Issue 11, November 2017