Malware Detection Through Decision Tree Classifier

Kamran Morovati, Sanjay Kadam

Abstract: ***The first part of this paper is devoted to a brief introduction, terminology and a comparison between different methods of preventing and detecting malware. The second portion of this paper presents a new method for classifying malicious files versus normal ones. Our approach is based on differences between assembly op-code frequencies in malware and benign classes. We have also utilized decision tree algorithms to simplify the classification.***

Keywords—Malware detection, Opcode frequencies, ANOVA test, Duncan multiple range test, Decision tree classifier,

I. Introduction

Malware incidents cost organizations and industries billions of dollars every year. In a 2012 worldwide survey on the financial impacts of malware, more than 2,600 business leaders and IT security practitioners were interviewed [1]. About 30% of participants thought a successful cyber-attack can cause damage between 200,001 and 300,000 U.S. dollars. Only 2 percent of respondents believed a single successful cyber-attack would cost their company less than 10,000 U.S. dollars. Malicious programs may seem like a relatively new concept, but they started appearing on dedicated networks such as ARPANET in the 1970s. In 1971, the Creeper virus appeared. It was able to replicate itself and its function was to display a simple message. The rabbit virus was another instance in that decade. The Rabbit virus spread across a network and generated copies of itself, impairing performance until a computer crashed. The term "Computer Virus" appeared in 1983 after Professor Len Adleman at Lehigh University demonstrated the concept at a seminar. In 1987, the Vienna Virus was introduced. It is the first virus that was able to destroy data. Fred Cohen published his first article about "Experiments with Computer Viruses" which were incorporated into his PhD thesis, "Computer Viruses - Theory and Experiments," published in 1986. His rather mathematically-oriented definition of a virus is still recognized today and does not encompass the negative connotation that the term virus has acquired nowadays [2].

Kamran Morovati
Computer Science Department, Pune University
IRAN
k.morovati@gmail.com

Sanjay Kadam, CDAC-Pune
Computer Science Department, Pune University
INDIA
sskadam@cdac.in

In 1990, the Chameleon virus emerged. It is known as the first polymorphic virus, which is able to change itself to avoid detection. In 1991 more than 300 viruses were documented and many antivirus products were introduced in the market. During 1996 malware like Win32.HLLP.DeTroi relayed data about compromised computers; thieves have stolen passwords and have been controlling systems remotely since then. Since the introduction of the web, E-mail and the Internet have become the primary transmission vectors, as scripts automatically load viruses from infected websites. In 2003 Slammer infected memory in computers worldwide through the Internet, clogging networks and causing shutdowns. Since 2007 Botnets have infected millions of Internet users worldwide. Zombie systems send spam and generate Denial of Service (DoS) attacks, compromising credentials and data. It is no wonder that today cyber security is the top concern of IT managers [3].

Malware Types and Terminology

Malware, an abbreviation for "Malicious Software," is software used or designed to disrupt or deny computer operations, gather sensitive information that leads to loss of privacy or exploitation, or gain access to private computer systems. It is a general term for any kind of hostile, intrusive, or annoying software that can appear in the form of code, scripts, active content, etc., and is able to infect a single computer, server, or an entire computer network [4]. Malware is usually categorized into "families" (referring to a particular type of malware with unique characteristics) and "variants" (usually a different version of code in a particular family). Malware is put in an information system to cause harm or to subvert that system for use in purposes other than intended by their owners [5]. Malware includes computer viruses, worms, Trojan horses, spyware, adware, rootkits, logic bombs, bots and other malicious programs. In law, malware is sometimes known as a computer contaminant [6]. Until a few years ago, viruses and worms were the most common types of malware, but nowadays other kinds have emerged and are extensively distributed.

Computer Viruses: Viruses are programs that replicate themselves. As soon as they execute, they make one or more copies of themselves. If these copies are also executed, they would reproduce even more copies. Usually a computer virus attaches itself to other executable files. This will ensure more efficient reproduction. Viruses must have two important parts to survive. Firstly, they must have a search subroutine to locate new files, disks, etc., in order to infect them. Secondly, they must have another part to copy the virus body effectively into the files which the search function locates. The most commonly used technique consisted in appending the viral code at the end of the executable file then modifying the entry

point to the point that viral code starts and finally resuming the execution of the normal code.

Worms: Worms are self-replicating programs that are able to replicate themselves, but unlike viruses, they do not infect other files or programs. They typically use unpatched vulnerabilities in network protocols and services such as email to distribute themselves quickly without user intervention. They also may use Auto-Run capability to propagate through digital medias like USB Pen drives, CDs/DVDs, etc. By consuming network bandwidth they may decrease the overall performance or sometimes clog the entire network.

Trojan Horses: Trojans are another type of malicious programs that masquerade as benign applications. Trojans use some appealing functions to lure users to execute them. They initially seem to perform desirable functions, but after execution, in addition to the expected functions, they steal data or damage the system. For example, a Trojan may appear like a PC game but in fact after being run, it may allow a hacker remote access to the victim's system over a network. Trojans could be used to install additional worms or viruses. These kinds of Trojans are known as Droppers. Backdoor (Trapdoor) Trojans also may be utilized by an intruder to gain a privileged remote access to the system by opening a TCP/UDP network port.

Logic Bombs: Typically, logic bombs are normal programs which contain some hidden malfunctions. These hidden malicious codes are intentionally inserted into the software and can be triggered automatically when pre-determined conditions are met. For example, a logic bomb could be activated on a specific date and time and it might delete particular file types or even execute another malware.

Rootkits: Rootkits are modified versions of popular programs, tools or operating system files that have been replaced with original ones by hackers as soon as breaking into a system. They are designed to conceal the existence of certain processes or programs from normal detection methods and they mostly provide continuous privileged access to the victim's computer. Rootkits are categorized in different classes depending on the level of operating system that they are running in. User-mode rootkits run along with other user's applications in the application ring (Ring 3), while Kernel-mode rootkits mainly run with the highest operating system privileges (Ring 0) by adding code or replacing operating system modules. The kernel-mode rootkits are often dangerous since they can alter the behavior of the operating system kernel. Therefore, it could be very difficult to detect them because of their ability to hide themselves from even kernel-level detector software.

Adware: An adware (advertising-supported software) automatically downloads banner ads and adverts pop-ups. In brief, any software that installs itself on the system without the user's knowledge and displays advertisements typically when the user browses the Internet is called as adware. Adware, by itself, is harmless; however, some may include integrated spyware such as key loggers and other privacy invasive software. Adware might be considered as a borderline case between malware and normal software and is also known as greyware [7].

Malware Detection Techniques

Nowadays, many anti-malware products from different professional companies are available in the market. These products utilize different techniques to combat malware. Anti-virus solutions are commonly installed at the operating system kernel level and generally consist of two fundamental modules: a database of information regarding virus signatures or common abnormal behaviors that viruses have and an inspection engine that utilizes the database for detection purposes.

Malware analysis methods can be categorized into two main groups:

a) Static Analysis

b) Dynamic Analysis

The first method is the analysis of malware without executing it. In this method, by collecting low level information such as Assembly Op-code Frequencies, which will be discussed later in this paper, Control Flow Graphs, System Call Graphs, Data Flow Graphs or other types of statistical data, we can analyze and detect malware. Many disassemblers and debuggers can be used to extract low level information. For instance, IDA Pro (Riesen and Bunker 2009) is a disassembler, which generates assembly language source code from machine-executable code. Static analysis is relatively fast and safe. It also produces few false positive errors. On the other hand, it has some disadvantages like the possibility of not detecting metamorphic and/or unknown viruses, which use code obfuscation techniques [8].

The second method is the analysis of suspected files during their execution. Dynamic analysis uses virtual or simulated environments, such as an emulator or a virtual machine, to monitor the behavior and functionality of executable files. The analysis environment must be invisible to the malware since the malware writer may use an anti-virtual machine or an anti-emulation tool to conceal their malware functions if they suspect that they are under analysis. Dynamic analysis fails to detect intended activities if the malware changes its behavior depending on trigger conditions such as the presence of a specific file or a specific day, as only a single execution path may be examined in each attempt [8].

Anti-virus software can operate in two different modes known as On-Demand and On-Access. On-Demand mode allows user to activate the antivirus manually at any desirable time, but On-Access mode automatically monitors system objects that programs or operating system access for any purpose. Table 1 summarizes the advantages and disadvantages of the malware detection methods.

Detection/Prevention Method	Advantage	Disadvantage
Signature Based	Fast, accurate, few false alarms	Not effective in the case of new unknown malware detection
Behavior Based	New malware detection	High false alarm, unproven
Code Emulation Based	Polymorphic/Metamorphic detection	Costly to implement
Integrity Checking	Simple, high detection rate on the file system	Slow, not preventative, high false alarms
Sandboxing	Damage preventive	Not compatible with all software

Table 1. Summary of the strength and weakness of malware detection techniques.

Malware Class	Number of Samples	File Size	Some Malware Families*
Virus	33	[10KB – 2 MB)	Win32/Tenga, Win32Neshta, Win32/Chir, Win32/Sality, etc.
Trojan	76	[10KB – 2 MB)	Win32/Ripinip, Win32/Wisdoor, Win32/Delf, Win32/SpyVoltar, etc.
Worm	45	[10KB – 1MB)	Win32/Mydoom.Q, Win32/Dabber, Win32/Bflient, Win32Pronny, etc.
Rootkit	15	[10KB – 5 MB)	Win32/Obfuscated.NSPWGMH, Win32/PSW, etc.
Adware	31	[10KB – 3 MB)	Win32/Adware.OneStep, Win32/Adware.Gamevance, Win32/Adware.GabPath, Win32/Adware.Filenolja, etc.

Table 2. Malware Samples Information (* ESET NOD32 Antivirus)

Proposed Methodology

In this section our proposed approach is described. Our method is based on the differences between opcode frequencies of the collected random samples of malicious and normal files. Our experiment consists of following steps:

Step 1: Malicious and Non-malicious sample files collection.
Step 2: Malware unpacking.
Step 3: Disassembling the binary executables to retrieve the assembly program.
Step 4: Extracting opcodes and calculating assembly function frequencies from the assembly program.
Step 5: Creating a database of observed results for each group of files and finding effective opcodes using the ANOVA and Duncan Multiple Range Test.
Step 6: Testing different decision tree algorithms to choose the best classifiers.

Sample Collection: The first step of our experiment consisted of collecting random samples of malware and normal files. To gather normal files which constitute our "Benign class", Portable Executable (PE) files were selected from two sources including the installed Cygwin software (a collection of tools which provide a Linux look and feel environment for Windows) and "System32" folder of MS-Windows 7 Ultimate version. From listed files, which yielded a normal distribution (based on file size), a total of 100 PE samples were selected and placed into four-size blocks, with at least 10 samples in each block. Regarding malware, five classes of interest including Viruses, Trojans, Adware, Worms and Rootkits were defined. Malware samples were collected from various online virus repositories such as the VX-Heaven website (the VX Heaven website at http://vx.netlux.org/index.html is unreachable since 23.03.2012 due to police investigation) and the Virus Sign website (http://www.virussign.com). Out of thousands of malicious files, a total number of 200 malware from different families were randomly chosen for further analysis. Table2 summarizes the malware samples' information.

MALWARE UNPACKING: AFTER OBTAINING MALWARE, THE NEXT STEP IS TO CHECK THE FILES FOR PACKING INFORMATION. PACKERS ARE MAINLY USED TO OBFUSCATE MALWARE SOURCE CODE OR TO COMPRESS THE EXECUTABLES. MALWARE DEVELOPERS USE PACKING TECHNIQUES AS A CHEAP AND EASY WAY OF TURNING A KNOWN PIECE OF MALWARE INTO SOMETHING NEW, WHICH MALWARE SCANNERS CAN'T DETECT. EXISTING COMMERCIAL MALWARE SCANNERS SEARCH BINARY FILES FOR PREDEFINED SIGNATURES, BUT OBFUSCATED MALWARE USES PACKERS TO PROTECT THEIR INTERNAL CODE AND DATA STRUCTURES. PACKERS COMPRESS AND ENCRYPT THE PE FILE IN THE SECONDARY MEMORY AND RESTORE THE ORIGINAL EXECUTABLE IMAGE WHEN LOADED INTO MAIN MEMORY (RAM) [9]. SOME MALWARE USES MULTIPLE PACKING TRANSFORMATIONS TOGETHER, WHICH MAKE THE AMOUNT OF WORK NECESSARY TO EMULATE THE FULL UNPACKING OPERATION MUCH MORE EXPENSIVE AND TIME CONSUMING. SOME PACKERS SHRINK FILE SIZE THROUGH COMPRESSION. FOR EXAMPLE, UPACK IS A WINDOWS-BASED COMPRESSION PACKER; IT COMPRESSES PE-FORMATTED FILES WITH VERY HIGH COMPRESSION RATES. MANY MALWARES HAVE USED IT TO AVOID DETECTION [10]. THE ULTIMATE PACKER FOR EXECUTABLES (UPX; HTTP://UPX.SOURCEFORGE.NET), AND ASPACK (WWW.ASPACK.COM) ARE SOME OTHER KNOWN EXAMPLES OF THIS GROUP. YODA'S CRYPTER (HTTP://YODAP.SOURCEFORGE.NET) AND POLYCRYPT PE (WWW.JLABSOFTWARE.COM) ARE POPULAR EXAMPLES OF THE CRYPTER PACKERS. PROTECTORS FEATURE BOTH COMPRESSORS AND CRYPTER PACKERS. SOME COMMERCIAL PROTECTORS, SUCH AS ARMADILLO (WWW.SILICONREALMS.COM) AND THEMIDA (WWW.OREANS.COM), ARE IN THIS GROUP. FINALLY, BOUNDLER PACKERS PACK MULTIPLE EXE FILES AND DATA FILES INTO ONE EXECUTABLE FILE. PEBUNDLE (WWW.BITSUM.COM/PEBUNDLE.ASP) AND MOLEBOX (WWW.MOLEBOX.COM) ARE SOME EXAMPLES OF THIS CLASS [10]. BY MEANS OF SOME NATIVE DEBUGGING TOOLS SUCH AS IDA PRO AND OLLYDBG (WWW.OLLYDBG.DE), WE CAN READ THE SECTION NAMES IN THE SECTION TABLE OF A PE HEADER, TO FIND OUT IF A PARTICULAR PE FILE IS PACKED OR NOT. SOME UNDERGROUND TOOLS ARE ALSO AVAILABLE, WHICH CAN AUTOMATICALLY EXTRACT PACKING INFORMATION OF A GIVEN PE FILE. IN OUR PROJECT SEVERAL TOOLS, SUCH AS THE PEID[11] AND THE EXEINFO-PE (HTTP://WWW.EXEINFO.ANTSERVE.COM),

WERE USED TO AUGMENT COMPILER AND PACKER INFORMATION TO THE DATABASE. PEID IS A DETECTOR USED FOR MOST COMMON PACKERS, CRYPTORS, COMPILERS AND EVEN SIGNATURE-BASED PACKER DETECTION IN PE FILES. THE RESULT OF ANALYZING OUR MALWARE USING PEID SHOWED THAT AROUND 63% OF SAMPLES WERE PACKED.

AFTER RETRIEVING THE PACKING INFORMATION OF OUR SAMPLES, WE USED APPROPRIATE UNPACKING TOOLS AVAILABLE ON THE WEB (E.G. WWW.WOODMANN.COM) TO ACQUIRE THE ORIGINAL CONTENT OF THE MALWARE.

Disassembly and analysis of malware: After checking all the malware and removing anomalies such as packers' obfuscation, we loaded the samples in to the de-facto industry standard disassembler, IDA Pro (6.1) [12]. This reliable disassembler translates binary content and generates assembly language source code from the machine-executable file format. It supports a variety of executable formats for different processors and operating systems.

Extracting Opcode Information: After loading each malware sample into the IDA Pro and running the InstructionCounter [13], which is a modified plugin of the IDA disassembler, we extract the assembly function statistics. For each malware, one text file was generated, which contained the frequency of used assembly functions in the corresponding binary file. Subsequently, these text files were imported to a MS-Excel spreadsheet and were augmented with the complete list of x86 instruction list available at [14], totaling 681 assembly functions. The benign files produced around 2 million opcodes in more than 130 different categories. Eighty-five opcodes accounted for more than 99.8% of opcodes found, 14 opcodes accounted for more than 91% and the top 5 opcodes accounted for 67.7% of extracted opcodes.

Malware samples yielded more than 7 million opcodes and more than 163 various assembly functions were found. Table 3 shows some descriptive statistics of the samples.

File Type	Total Opcodes	Opcodes found	Top 14 Opcodes
Goodware	19798076	>130	mov(48%), call(7%), jmp(4%), cmp(4%), jz(3%), test(3%), lea(3%), push(3%), add(3%), pop(3%), sub(2%), jnz(2%), movzx(1%), retn(1%)
Viruses	932016	>141	mov(23%), push(20%), call(10%), pop(10%) , cmp(4%), lea(4%), jz(4%), jmp(3%), test(3%) , jnz(3%), add(3%), xor(2%), retn(2%), sub(1%)
Trojans	3083467	>146	mov(28%), push(16%), call(9%), pop(6%), lea(4%), add(4%), cmp(4%), jmp(3%), jz(3%), xor(3%), sub(2%),test(2%), retn(2%), jnz(2%)
Worms	1043123	>133	mov(26%), push(21%), call(12%), lea(8%), pop(4%), cmp(4%), add(3%), jz(3%), jmp(3%), test(2%), jnz(2%), xor(2%), retn(2%), sub(1%)
Ad-ware	1614391	>163	mov(24%), push(17%), call(9%), pop(6%), cmp(6%), jz(4%), lea(4%), test(3%), jnz(3%), add(3%), jmp(3%), xor(2%), retn(2%), sub(1%)
Rootkits	720072	100	mov(26%), push(19%), call(9%),pop(6%), nop(4%), lea(4%), cmp(3%), add(3%), jz(3%), xor(2%), jmp(2%), retn(2%), test(2%), jnz(2%)

Table 3. Descriptive statistics of samples.

ANOVA and Duncan Multiple Range Tests: According to [41], more than 680 assembly opcodes are defined to date, but our observation revealed that only less than 200 opcodes are in use by our samples. In a Microsoft Excel spread sheet, a 300 X 681 contingency table was designed (rows: samples of different classes, columns: opcode frequencies). In this phase we want to detect the effective opcodes and reduce the number of assembly functions that we are going to use as inputs of our classifier later. We also want to know if there is a statistically significant difference in opcode frequencies between software classes. The null-hypothesis and the alternative hypothesis were defined as following:

H0: Statistically, there is no significant difference between the opcode frequencies.

H1: Statistically, There are some associations between opcodes.

To determine the effectiveness and the significance of the opcodes, we used "ANOVA" (Analysis of variance) and "Duncan Multiple Range" tests. Since we have more than two variables, the ANOVA test was chosen to compare the variances of the samples. Afterwards the Duncan multiple range test was used to segment different treatments (i.e. opcodes), which more or less has same significance according to the differences in mean values. In statistics, Duncan's multiple range test (MRT) is a multiple comparison procedure developed by David B. Duncan in 1955. Duncan's MRT belongs to the general class of multiple comparison procedures that use the Studentized range statistic to compare sets of means. Before using the Duncan test we must ensure that there is statistically significant difference between variables frequencies. Analysis of variance (ANOVA) is a collection of statistical models, and their associated procedures, in which the observed variance in a particular variable is partitioned into components attributable to different sources of variation. ANOVA provides a statistical test of whether or not the means of multiple groups are all equal, and therefore it generalizes t-test to more than two groups. Doing multiple two-sample t-tests would result in an increased chance of committing a type I error. Therefore, ANOVA is useful in comparing several means. To analyze the variance of samples, all data from the Excel spread sheet was imported into IBM SPSS software and tests were run. Table 4 shows the result of the ANOVA test.

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected model	42.483a	680	.062	1441.358	.000
Intercept	.418	1	.418	9640.073	.000
Opcodes	42.483	680	.062	1441.358	.000
Error	8.383	193404	4.334E-5		
Total	51.283	194085			
Corrected Total	50.865	194084			

a. R Squared = .835 (Adjusted R Squared = .835)

Table 4. ANOVA test results.

Result of the ANOVA test indicates that the significant level is less than 0.05, which rejects the null-

hypothesis (H_0) and confirms the alternative hypothesis (H_A). After observing the results of the ANOVA test, we run Duncan test to know the most significant and the least significant opcode groups. Generally the Duncan test is based on the idea that the means must be compared according to the variable range. In our experiment, the Duncan test grouped the opcodes into 27 segments based on their significance. In this step we proved that opcodes are good malware predictors because statistically there are meaningful differences in opcode frequencies between samples of dissimilar classes. In addition, Duncan test helped us to segment opcodes. In this segmentation, functions belonging to each group have same significance, hence, practically only one member function is enough to represent the entire belonging segment. This simply reduces number of the effective opcodes from 681 to 27. Later, we use these 27 functions as inputs of our classifier. Table 5 shows the list of this selected opcodes. Each opcode is chosen from different segment so that it represents the entire belonging group.Table 5, summarizes the segments created by the Duncan multiple range test.

Segment#	Selected Opcode
1	cmovnz(0.000123158)
2	std(0.000135789)
3	div(0.000376842)
4	fldcw(0.000315088)
5	cld(0.000332982)
6	adc(0.000443860)
7	cdq(0.000472281)
8	jns(0.000595789)
9	js(0.000778246)
10	neg(0.000971930)
11	fld(0.001556140)
12	imul(0.001677895)
13	nop(0.003014035)
14	or(0.006754035)
15	movzx(0.008841053)
16	inc(0.009570526)
17	retn(0.018785614)
18	sub(0.019944211)
19	jnz(0.022636842)
20	test(0.029306316)
21	add(0.032072281)
22	jz(0.034317895)
23	cmp(0.04137473)
24	pop(0.045806667)
25	call(0.092000351)
26	push(0.12995614)
27	mov(0.33630701)

Table 5. Selected opcodes through the Duncan test

CLASSIFICATION OF THE SAMPLES: THE LAST PHASE OF OUR EXPERIMENT IS TO DESIGN A MACHINE CLASSIFIER TO IDENTIFY MALWARE VERSUS GOODWARE. THIS CLASSIFICATION IS BASED ON DISSIMILARITIES OF THE OPCODE FREQUENCIES. IN THIS STEP, BY USING MACHINE LEARNING TECHNIQUES SUCH AS THE DECISION TREES, WE TRIED TO AUTOMATE THE CLASSIFICATION TASK.

WE DID TWO SEPARATE CLASSIFICATIONS; FIRST WE DEFINED TWO DESIRED CLASS LABELS (BENIGN AND MALWARE) AND TESTED DIFFERENT ALGORITHMS ON OUR SAMPLES. SECONDLY WE DEFINED 6 DESIRED CLASS LABELS (BENIGN, VIRUS, TROJAN, WORM, ROOTKIT AND ADWARE) AND AGAIN WE DID THE CLASSIFICATION. IN BOTH SCENARIOS THE INPUTS OF CLASSIFIERS WERE THE FREQUENCIES OF OPCODES LISTED IN TABLE 5.

decision tree classifier

A DECISION TREE IS A CLASSIFICATION TOOL THAT USES A TREE-LIKE GRAPH OR MODEL OF DECISIONS AND THEIR POSSIBLE CONSEQUENCES. A DECISION TREE IS A FLOWCHART-LIKE TREE STRUCTURE, WHERE EACH INTERNAL NODE (NON-LEAF NODE) DENOTES A TEST ON AN ATTRIBUTE, EACH BRANCH REPRESENTS AN OUTCOME OF THE TEST, AND EACH LEAF NODE (OR TERMINAL NODE) HOLDS A CLASS LABEL. THE TOPMOST NODE IN A TREE IS THE ROOT NODE [15]. A DECISION TREE BUILT FROM A TRAINING SET CAN LATER BE USED TO CLASSIFY TRAINING AND RECALL PATTERNS. GIVEN A TUPLE X FOR WHICH THE ASSOCIATED CLASS LABEL IS UNKNOWN, THE ATTRIBUTE VALUES OF THE TUPLE ARE TESTED AGAINST THE DECISION TREE. A PATH IS TRACED FROM THE ROOT TO A LEAF NODE, WHICH HOLDS THE CLASS PREDICTION FOR THAT TUPLE. DECISION TREES CAN EASILY BE CONVERTED TO CLASSIFICATION RULES. DECISION TREE IS ONE OF THE MOST POPULAR CLASSIFICATION AND DECISION SUPPORT TOOLS. SOME OF ITS ADVANTAGES ARE AS FOLLOWING: [15]

- TO CONSTRUCT A DECISION TREE THERE IS NO NEED OF PARAMETER SETTING.
- DECISION TREES CAN HANDLE HIGH DIMENSIONAL DATA.
- REPRESENTATION OF ACQUIRED KNOWLEDGE IN TREE FORM IS SIMPLE AND EASY TO ASSIMILATE BY HUMANS.
- SIMPLE AND FAST LEARNING AND CLASSIFICATION PHASES.
- HIGH CLASSIFICATION ACCURACY.
- ABLE TO HANDLE BOTH NUMERICAL AND CATEGORICAL DATA.

TO CLASSIFY OUR SAMPLES USING DECISION TREES, WE CONVERTED OUR EXCEL DATABASE INTO THE CSV (COMMA-SEPARATED VALUES) FORMAT AND THEN WE IMPORTED THIS DATA FILE INTO THE WEKA [16] SOFTWARE. WEKA IS A COLLECTION OF MACHINE LEARNING ALGORITHMS FOR DATA MINING TASKS AND IT CONTAINS TOOLS FOR DATA PRE-PROCESSING, CLASSIFICATION, REGRESSION, CLUSTERING, ASSOCIATION RULES, AND VISUALIZATION. IT IS ALSO WELL-SUITED FOR DEVELOPING NEW MACHINE LEARNING SCHEMES [16].

WE TESTED VARIOUS DECISION TREE LEARNING ALGORITHMS, WHICH ARE AVAILABLE IN THE TREE CLASSIFIER PART OF THE WEKA SOFTWARE. THESE ALGORITHMS INCLUDE J48[17], J48-GRAFT[18], BFTREE[19], FT[20], LADTREE[21], LMT[22], NBTREE[23], RANDOM FORESTS [24] AND RANDOMTREE, REPTREE[25], SIMPLECART[26] AND DECISIONSTUMP[27].

ACCORDING TO OUR OBSERVATION, THE NB TREE AND RANDOM FOREST METHODS ACHIEVED THE BEST ACCURACY WITH MORE THAN 98% CLASSIFICATION SUCCESS RATE. TABLE 6 SHOWS THE CLASSIFICATION INFORMATION OF THESE TWO ALGORITHMS. TABLE 7 AND TABLE 8 ALSO GIVE MORE DETAILS REGARDING THESE TWO METHODS.

	Random Forest	NB Tree
Total Number of Instances	300	300
Correctly Classified Instances	295	295
(%)	98.33%	98.33%
Incorrectly Classified Instances	5	5
(%)	1.66%	1.66%
Kappa statistic	0.9622	0.9622
Mean absolute error	0.0399	0.0192
Root mean squared error	0.1328	0.1273
Relative absolute error	9.0114%	4.3406%
Root relative squared error	28.2375%	27.08%

Table 6. Results of classification with two class labels using Random Forest and NBTree algorithms.

Class Label	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Malware	0.99	0.03	0.985	0.99	0.988	0.991
Benign	0.97	0.01	0.98	0.97	0.975	0.991
Weighted Avg.	0.983	0.024	0.983	0.983	0.983	0.991

Table 7. Detailed Accuracy Information of NBTree and Random Forest algorithm.

Classified As ➔	Malware	Benign
Malware	199	2
Benign	3	96

TABLE 8. CONFUSION MATRIX OF NBTREE AND RANDOM FOREST CLASSIFIERS

Table 9, summarizes the experiment outcomes for the classification with six class labels including: Virus, Trojan, Adware, Worm, Rootkit and Benign. In this case, again the Random Forest algorithm yields the best accuracy with 79.66% success rate. Table 10 recapitulates the detailed accuracy information of the Random Forest algorithm and table 11 shows the respective confusion matrix.

	Random Forest
Total Number of Instances	300
Correctly Classified Instances	239
(%)	79.66%
Incorrectly Classified Instances	61
(%)	20.33%
Kappa statistic	0.7372
Mean absolute error	0.0924
Root mean squared error	0.22
Relative absolute error	35.55%
Root relative squared error	61.07%

Table 9. Random Forest algorithm result.

Class Label	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Virus	0.606	0.045	0.625	0.606	0.615	0.888
Adware	0.688	0.034	0.71	0.688	0.698	0.919
Worm	0.622	0.039	0.737	0.622	0.675	0.864
Benign	0.98	0.035	0.933	0.98	0.956	0.984
Trojan	0.776	0.094	0.738	0.776	0.756	0.932
Rootkit	0.867	0.007	0.867	0.867	0.867	0.923
Weighted Avg.	0.797	0.05	0.793	0.797	0.794	0.932

Table 10. Detailed Accuracy Information of the Random Forest Decision Tree Learning algorithm.

Classified As ➔	Virus	Adware	Worm	Benign	Trojan	Rootkit
Virus	20	3	5	0	5	0
Adware	3	22	0	0	7	0
Worm	5	2	28	4	6	0
Benign	0	0	1	97	1	0
Trojan	4	4	4	3	59	2
Rootkit	0	0	0	0	2	13

Table 11. Confusion Matrix of Random Forest Classifier.

Conclusion and Future Work

In this paper we have presented a new method for differentiating between malware and benign files by means of decision tree classification technique. The inputs of our classifier were the opcode frequencies of our samples. The classification success rate was more than 98% which demonstrates the reliability of our proposed approach.

Instead of Opcode frequencies, other static features such as the API function calls can be used for the same purpose. In this research, the Duncan test is used to reduce the number of classification inputs. Other attribute selection methods may reduce inputs even more. Rather than decision tree, other classification techniques can be employed too.

In another research, we repeated the experiment and we used different classification methods such as ANN, SVM and Naive Bayes classifiers. We are going to publish the results of this trial too.

ACKNOLEDGMENT

A special thank of ours goes to Dr. Daniel Bilar from Wellesley College (MA, USA) who helped us regarding malware collection. His previous work [28] made our research easier. We also wish to gratefully thank Mrs. Amy Da Siilva for her editing assistance.

REFERENCES

[1] Financial Damage Caused by Cyber Attacks in the United States. http://www.statista.com/statistics/193444/financial-damage-caused-by-cyber-attacks-in-the-us/

[2] Fred Cohen, PhD thesis: "Computer Viruses - Theory and Experiments", 1984. http://all.net/books/virus/index.

[3] Samuel Greengard, A Brief History of Malware, 2010-03-17.

[4] Robert Moir, Defining Malware. http://technet.microsoft.com/en-us/library/dd632948.aspx

[5] ITU Study on the Financial Aspects of Network Security: Malware and Spam, Final Report July 2008. www.itu.int/ITU-D/cyb/cybersecurity/docs/itu-study-financial-aspects-of-malware-and-spam.pdf

[6] USA National Conference of State Legislatures. http://www.ncsl.org/issues-research/telecom/state-virus-and-computer-contaminant-laws.aspx

[7] Tulloch, Mitch (2003). Koch, Jeff; Haynes, Sandra. eds. Microsoft Encyclopedia of Security. Redmond, Washington: Microsoft Press. p. 16. ISBN 0-7356-1877-1.

[8] Egele, M., T. Scholte, E. Kirda and C. Kruegel, A survey on automated dynamic malware analysis techniques and tools, 2011. ACM Comput. Surv., 5: 1- 49

[9] Sharif, M.; Yegneswaran, V.; Saidi, H.; Porras, P. & Lee, W., "Eureka: A framework for enabling static malware analysis", Computer Security - ESORICS, Lecture Notes in Computer Science LNCS, Springer, 008, 5283/2008, 481-500.

[10] Wei Yan, Zheng Zhang, Nirwan Ansari, "Revealing Packed Malware", Secure Systems, Published by the IEEE Computer Society, Volume: 6 , Issue:5, PP: 65 – 69, ISSN : 1540-7993, Sept. 2008

[11] Snaker et al, PEiD, V 0.95. http://www.woodmann.com/collaborative/tools/index.php/Category:Packer_Identifiers, 2008.

[12] IDA Pro Dissasember, Hex-Rays, An Advanced Interactive Multiprocessor Disassembler, http://www.hex-rays.com, October, 2012.

[13] Porst, S., InstructionCounter v.1.02 (IDA Plugin), 2006, http://www.the-interweb.com/serendipity/index.php?/archives/62-IDA-InstructionCounter-plugin-1.02.html

[14] Wikipedia, x86 instruction listings, Accessed May 2012, http://en.wikipedia.org/wiki/X86_instruction_listings

[15] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, 2nd edition, Publisher: Morgan-Kufmann ISBN 13: 978-1-55860-901-3

[16] WEKA, The University of Waikato, http://www.cs.waikato.ac.nz/~ml/weka/

[17] Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

[18] Geoff Webb: Decision Tree Grafting From the All-Tests-But-One Partition. In: , San Francisco, CA, 1999.

[19] Haijian Shi (2007). Best-first decision tree learning. Hamilton, NZ.

[20] Niels Landwehr, Mark Hall, Eibe Frank (2005). Logistic Model Trees.

[21] Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, Eibe Frank, Mark Hall: Multiclass alternating decision trees. In: ECML, 161-172, 2001.

[22] Niels Landwehr, Mark Hall, Eibe Frank (2005). Logistic Model Trees. Machine Learning. 95(1-2):161-205.

[23] Ron Kohavi: Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In: Second International Conference on Knowledge Discovery and Data Mining, 202-207, 1996.

[24] Leo Breiman (2001). Random Forests. Machine Learning. 45(1):5-32.

[25] REPTree Algorithm, http://weka.sourceforge.net/doc/weka/classifiers/trees/REPTree.html

[26] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone (1984). Classification and Regression Trees. Wadsworth International Group, Belmont, California.

[27] Decision Stump Algorithm, http://weka.sourceforge.net/doc/weka/classifiers/trees/DecisionStump.html

[28] Bilar, D. (2007) "opcodes as predictor for malware", Int. J. Elctronic Security and Digital Forensics, Vol.1, No.2 , pp. 156-168.

About Authors:

Kamran Morovati is a senior network administrator and information security researcher. He is a Ph.D candidate (computer science) in Department of Computer Science–University of Pune. He has a M. Tech. in Information Technology from Amirkabir University of technology (Tehran Polytechnic) and BE in software engineering from Samsipour University of Technology (Tehran). His research interests include Information Security, Computer Networks, Soft Computing and Data Mining.

Dr. Sanjay Kadam works as a Joint Director in the Evolutionary Computing and Image Processing Group at C-DAC, Pune. He has a M. Sc. in Mathematics from Pune University, an M.Tech in Computer Science from IIT, New Delhi, and a Ph.D in Computer Science from the University of London. His research interests include Image Processing, Parallel Processing, Neural Networks, and Soft computing.