Malware Classification Challenge

Palak Agarwal[1] and Karan Bansal[1]

Indian Institute of Technology, Kanpur, India

Abstract. In this challenge, we aim to classify malware into families by using machine learning techniques. Our dataset is very large, having 10,800 malware files each for training and test set. These files are in .bytes and .asm format and our challenge is to find the characteristic features in these files that would help us in distinguishing files belonging to different families from each other. Not all features we used turned out to be good and finding the best possible combination of all the features available is the trickier task here. We will show you, through our results, that by adding some features to an old feature set might worsen our results . . .

Keywords: Malware analysis, Polymorphism, Data mining

1 Motivation

Malware pose a great challenge in our day-to-day life. With their ever increasing number, it has become absolutely necessary for us to find an efficient method to get rid of them. The effectiveness of the existing anti-malware has reduced significantly after the introduction of polymorphism in the computer world. Polymorphism encrypts the code of the viruses, thus changing their signatures too. To understand how this affects the whole working of anti-malware softwares, you need to know how they work, anti-malware have a database of virus signatures, which gets updated regularly and whenever they encounter a file, they check if the signature of this file is in their database or not[6]. If it is, then it is treated as a virus, otherwise it is treated as a clean file. Polymorphic viruses have a snippet of code within them, which keeps on changing their code, without altering the functioning of the virus. This changes the virus's signature too and they are no longer detected by the anti-malware. This calls for a better approach to classify malware.

1.1 Other works

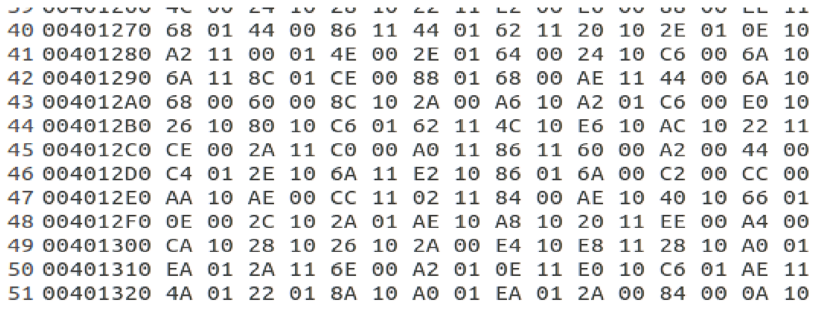
It has been shown that static analysis of virus is more effective than dynamic analysis[8]. Much work has been done on the malware problem and by using varied techniques.[7] have used image processing technique to classify malware into families and were able to get a good accuracy.[4], [3] uses dynamic analysis and claim to have pretty good results. [8] shows that dynamic analysis is not feasible for a large dataset like ours, thus limiting us to only use static analysis techniques. We are not using image processing techniques as the images are formed using the .byte files and so, whatever features have been applied in the image processing techniques, similar features can be applied in the bytes file.

2 Methodology

To solve the problem, we had to extract relevant features from the dataset that was given to us by Kaggle [2]. This was the non-trivial part of the challenge to identify those features that would help us to classify these files.

2.1 Features used

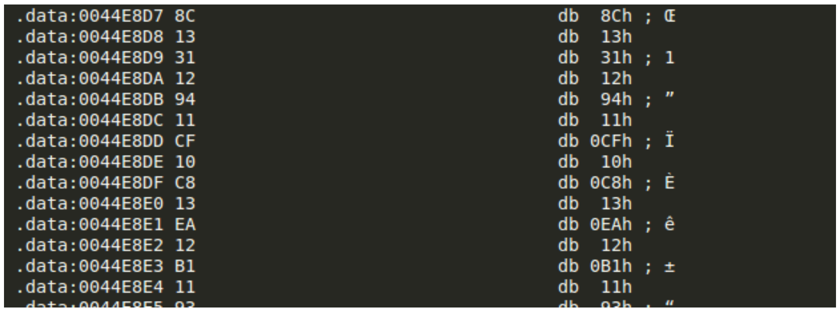
1. We started by taking the normalised frequency of the two digit hexadecimal values in our bytes file as the feature. Since each hexadecimal digit can take 16 values each, we had 256 possible values and so we had 256 features in this case. The frequency of these 256 values was counted for each .bytes file in the dataset and normalised.



```
00401270 68 01 44 00 86 11 44 01 62 11 20 10 2E 01 0E 10
00401280 A2 11 00 01 4E 00 2E 01 64 00 24 10 C6 00 6A 10
00401290 6A 11 8C 01 CE 00 88 01 68 00 AE 11 44 00 6A 10
004012A0 68 00 60 00 8C 10 2A 00 A6 10 A2 01 C6 00 E0 10
004012B0 26 10 80 10 C6 01 62 11 4C 10 E6 10 AC 10 22 11
004012C0 CE 00 2A 11 C0 00 A0 11 86 11 60 00 A2 00 44 00
004012D0 C4 01 2E 10 6A 11 E2 10 86 01 6A 00 C2 00 CC 00
004012E0 AA 10 AE 00 CC 11 02 11 84 00 AE 10 40 10 66 01
004012F0 0E 00 2C 10 2A 01 AE 10 A8 10 20 11 EE 00 A4 00
00401300 CA 10 28 10 26 10 2A 00 E4 10 E8 11 28 10 A0 01
00401310 EA 01 2A 11 6E 00 A2 01 0E 11 E0 10 C6 01 AE 11
00401320 4A 01 22 01 8A 10 A0 01 EA 01 2A 00 84 00 0A 10
```

A part of .bytes file of one of the malware

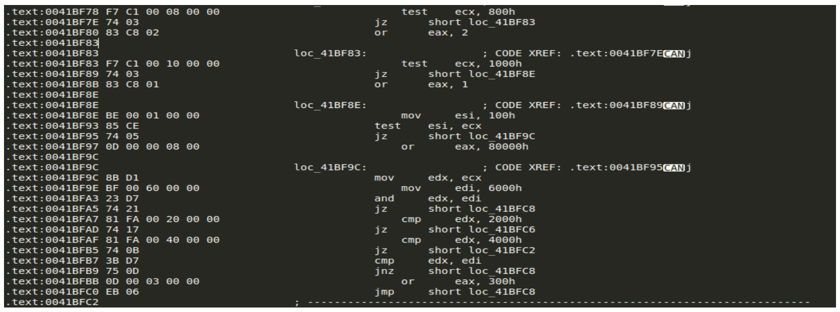
2. A similar feature was used for the asm files, the only difference being that there were specific places in the asm files, where the frequency was counted. In the given image, we are counting the hex values in second column of each row (8C,13,31,12,94,etc.)



```
.data:0044E8D7 8C                              db  8Ch ; Œ
.data:0044E8D8 13                              db  13h
.data:0044E8D9 31                              db  31h ; 1
.data:0044E8DA 12                              db  12h
.data:0044E8DB 94                              db  94h ; ”
.data:0044E8DC 11                              db  11h
.data:0044E8DD CF                              db 0CFh ; Ï
.data:0044E8DE 10                              db  10h
.data:0044E8DF C8                              db 0C8h ; È
.data:0044E8E0 13                              db  13h
.data:0044E8E1 EA                              db 0EAh ; ê
.data:0044E8E2 12                              db  12h
.data:0044E8E3 B1                              db 0B1h ; ±
.data:0044E8E4 11                              db  11h
```

A part of .asm file of one of the malware

3. Next, we used the frequency of different assembly functions in the asm files. We used instructions like ’jump’, ’push’, ’pop’, ’add’, ’dec’ and some others. 10 such functions were taken as our features.



```
.text:0041BF78 F7 C1 00 08 00 00                      test    ecx, 800h
.text:0041BF7E 74 03                               jz      short loc_41BF83
.text:0041BF80 83 C8 02                            or      eax, 2
.text:0041BF83
.text:0041BF83                     loc_41BF83:                  ; CODE XREF: .text:0041BF7ECANj
.text:0041BF83 F7 C1 00 10 00 00                      test    ecx, 1000h
.text:0041BF89 74 03                               jz      short loc_41BF8E
.text:0041BF8B 83 C8 01                            or      eax, 1
.text:0041BF8E
.text:0041BF8E                     loc_41BF8E:                  ; CODE XREF: .text:0041BF89CANj
.text:0041BF8E BE 00 01 00 00                         mov     esi, 100h
.text:0041BF93 85 CE                               test    esi, ecx
.text:0041BF95 74 05                               jz      short loc_41BF9C
.text:0041BF97 0D 00 00 08 00                         or      eax, 80000h
.text:0041BF9C
.text:0041BF9C                     loc_41BF9C:                  ; CODE XREF: .text:0041BF95CANj
.text:0041BF9C 8B D1                               mov     edx, ecx
.text:0041BF9E BF 00 60 00 00                         mov     edi, 6000h
.text:0041BFA3 23 D7                               and     edx, edi
.text:0041BFA5 74 21                               jz      short loc_41BFC8
.text:0041BFA7 81 FA 00 20 00 00                      cmp     edx, 2000h
.text:0041BFAD 74 17                               jz      short loc_41BFC6
.text:0041BFAF 81 FA 00 40 00 00                      cmp     edx, 4000h
.text:0041BFB5 74 0B                               jz      short loc_41BFC2
.text:0041BFB7 3B D7                               cmp     edx, edi
.text:0041BFB9 75 0D                               jnz     short loc_41BFC8
.text:0041BFBB 0D 00 03 00 00                         or      eax, 300h
.text:0041BFC0 EB 06                               jmp     short loc_41BFC8
.text:0041BFC2                     ; ---------------------------------------------------------------------------
```

A part of .asm file of one of the malware
Functions like jz, add, push can be seen in the image

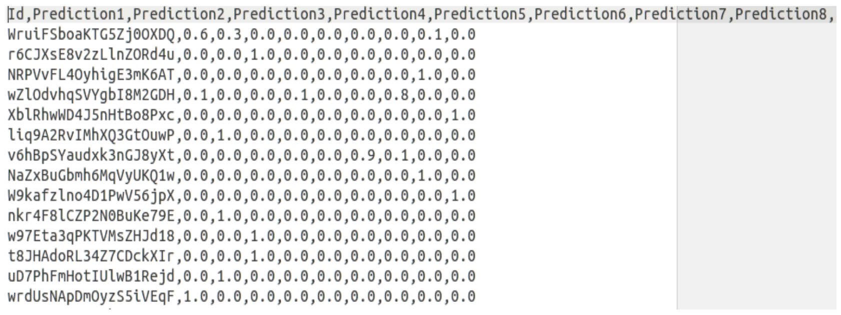
4. We also used other features like size of the files, Shannon entropy[5] and a mixture of the above features.We used the function of Shannon Entropy over the frequency of various bytes to get an idea of randomness of the files.

2.2 Classifiers used

We have only used 2 classifiers, Random Forest and Support Vector Machines, for the project, as this was not the main focus of the challenge. Random forest were majorly used and gave decent results[9]. We varied the number of trees to get good results on a given feature set.

3 Results

To check the validity of our classifier, it is applied on the test set to get the probabilities of the malware file belonging to each of the 9 families. The prediction file, shown below, gives a brief idea of this.



```
Id,Prediction1,Prediction2,Prediction3,Prediction4,Prediction5,Prediction6,Prediction7,Prediction8,
WruiFSboaKTG5Zj0OXDQ,0.6,0.3,0.0,0.0,0.0,0.0,0.0,0.1,0.0
r6CJXsE8v2zLlnZORd4u,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0
NRPVvFL4OyhigE3mK6AT,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0
wZlOdvhqSVYgbI8M2GDH,0.1,0.0,0.0,0.1,0.0,0.0,0.8,0.0,0.0
XblRhwWD4J5nHtBo8Pxc,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0
liq9A2RvIMhXQ3GtOuwP,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
v6hBpSYaudxk3nGJ8yXt,0.0,0.0,0.0,0.0,0.0,0.9,0.1,0.0,0.0
NaZxBuGbmh6MqVyUKQ1w,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0
W9kafzlno4D1PwV56jpX,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0
nkr4F8lCZP2N0BuKe79E,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
w97Eta3qPKTVMsZHJd18,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0
t8JHAdoRL34Z7CDckXIr,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0
uD7PhFmHotIUlwB1Rejd,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
wrdUsNApDmOyzS5iVEqF,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
```

3.1 Evaluation

The score in the competition is calculated using the multi-class logarithmic loss function, given by:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \tag{1}$$

Our goal is to take the score as close to 0 as possible.

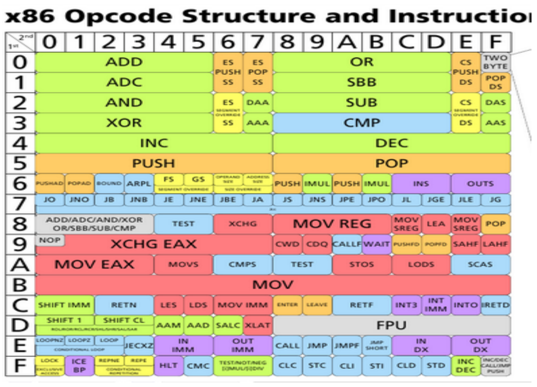
Our best score on the test board is 0.0.082228293, which uses the frequency of the 256 hex values as its features and uses random forest classifier.
By using the other techniques, we have got the following results:

Table 1. An overview of the submissions

Feature	Score
256 hex values in bytes file(256 trees)	0.082228293
256 normalised hex values in bytes file(500 trees)	0.082421290
1st feature + size of the file	0.099343705
Shannon entropy	14.111708334
10 functions & 30 trees	0.140188703
Size of the files	5.610533917
14 functions & 10 trees	0.156611540
10 functions & 10 trees	0.153125351
256 hex values & 10 trees (.bytes)	0.192934515
256 hex values (.asm)	0.250228987

3.2 Analysis

1. Frequency of hexadecimal values in the bytes file is actually equivalent to counting the frequency of the functions in the asm files. The functions used in the malware make a lot of difference and so, it turned out to be a good feature. You can see the correlation by this figure.



x86 Opcode Structure and Instructio

1st \ 2nd	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD						ES PUSH	ES POP	OR						CS PUSH	TWO BYTE
1	ADC						SS	SS	SBB						DS	POP DS
2	AND						ES SEGMENT OVERRIDE	DAA	SUB						CS SEGMENT OVERRIDE	DAS
3	XOR						SS	AAA	CMP						DS	AAS
4	INC								DEC							
5	PUSH								POP							
6	PUSHAD	POPAD	BOUND	ARPL	FS SEGMENT OVERRIDE	GS	OPERAND SIZE	ADDRESS SIZE	PUSH	IMUL	PUSH	IMUL	INS		OUTS	
7	JO	JNO	JB	JNB	JE	JNE	JBE	JA	JS	JNS	JPE	JPO	JL	JGE	JLE	JG
8	ADD/ADC/AND/XOR OR/SBB/SUB/CMP				TEST		XCHG		MOV REG				MOV SREG	LEA	MOV SREG	POP
9	NOP	XCHG EAX							CWD	CDQ	CALLF	WAIT	PUSHFD	POPFD	SAHF	LAHF
A	MOV EAX				MOVS		CMPS		TEST		STOS		LODS		SCAS	
B	MOV															
C	SHIFT IMM		RETN		LES	LDS	MOV IMM		ENTER	LEAVE	RETF		INT3	INT IMM	INTO	IRETD
D	SHIFT 1 ROL/ROR/RCL/RCR/SHL/SHR/SAL/SAR		SHIFT CL		AAM	AAD	SALC	XLAT	FPU							
E	LOOPNZ CONDITIONAL LOOP	LOOPZ	LOOP	JECXZ	IN IMM		OUT IMM		CALL	JMP	JMPF	JMP SHORT	IN DX		OUT DX	
F	LOCK EXCLUSIVE ACCESS	ICE BP	REPNE CONDITIONAL REPETITION	REPE	HLT	CMC	TEST/NOT/NEG [I]MUL/[I]DIV		CLC	STC	CLI	STI	CLD	STD	INC DEC	INC/DEC CALL/JMP PUSH

Image taken from [1]

2. Frequency of hexadecimal values at specific positions in the asm file did not give a nice result. These values do not have any meaning related to them and were used because for the few files for which their frequency was checked, gave pretty good values.

3. Frequency of specific assembly language instructions. Some functions are more important than others. So, the same should be true for the viruses and hence, their frequency should have been a good feature but it turns out it is not.

4. The size of the files alone could not have been a very good feature, because the damage that a virus can cause is independent of its size. The function that a virus performs is mostly independent of its size and is verified by the results.

5. The number of trees used in Random forest classifier makes a lot of difference as evident from the results. Using less trees can give you bad results like in the case of 256 hex values for the bytes file. It give a significantly good result when we increased the number of trees to 256.

6. A clean file was also analyzed by the classifier and it gave a probability of 0.57 for one of the malware families while for most of our test set files, the probability of one of the families is greater than 0.8. So, our classifier can also differentiate between clean files and malware.

4 Conclusions

The challenge posed by the problem is not completely solved. To find the feature that can completely distinguish these files into families is still our major concern. Though we did get pretty close to 0, by getting a score of 0.082 but we were not able to achieve perfect 0. The method used to implement it does not make much of a difference and so, our results will get better only if we find the best, relevant features. The analysis of the result tells us that hex values(of the byte file), functions(in the asm files) are good features. We need to find more features and patterns to get better results.

5 Code

Base code for this project is taken from Vishnu Chevli (github.com/vrajs5/Microsoft-Malware-Classification-Challenge). The code used for this project is available at https://bitbucket.org/palakag/malware-analysis-code/overview

References

1. Opcode image.
2. Microsoft malware classification challenge, 2015.
3. Michael Bailey, Jon Oberheide, Jon Andersen, Z.Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. In Christopher Kruegel, Richard Lippmann, and Andrew Clark, editors, *Recent Advances in Intrusion Detection*, volume 4637 of *Lecture Notes in Computer Science*, pages 178–197. Springer Berlin Heidelberg, 2007.
4. Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering.
5. Gregory Conti, Sergey Bratus, Anna Shubina, Benjamin Sangster, Roy Ragsdale, Matthew Supan, Andrew Lichtenberg, and Robert Perez-Alemany. Automated mapping of large binary objects using primitive fragment type classification. *Digital Investigation*, 7, Supplement(0):S3 – S12, 2010. The Proceedings of the Tenth Annual {DFRWS} Conference.
6. Eric Filiol. Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology*, 2(1):35–50, 2006.
7. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, pages 4:1–4:7, New York, NY, USA, 2011. ACM.
8. Lakshmanan Nataraj, Vinod Yegneswaran, Phil Porras, and Jian Zhang. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Workshop on Artificial Intelligence and Security (AISec)*, Oct 2011.
9. Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee. Detecting internet worms using data mining techniques. *Journal of Systemics, Cybernetics and Informatics*, pages 48–53, 2009.