SMA-2020

Android Malware Family Classification using Images from Dex Files

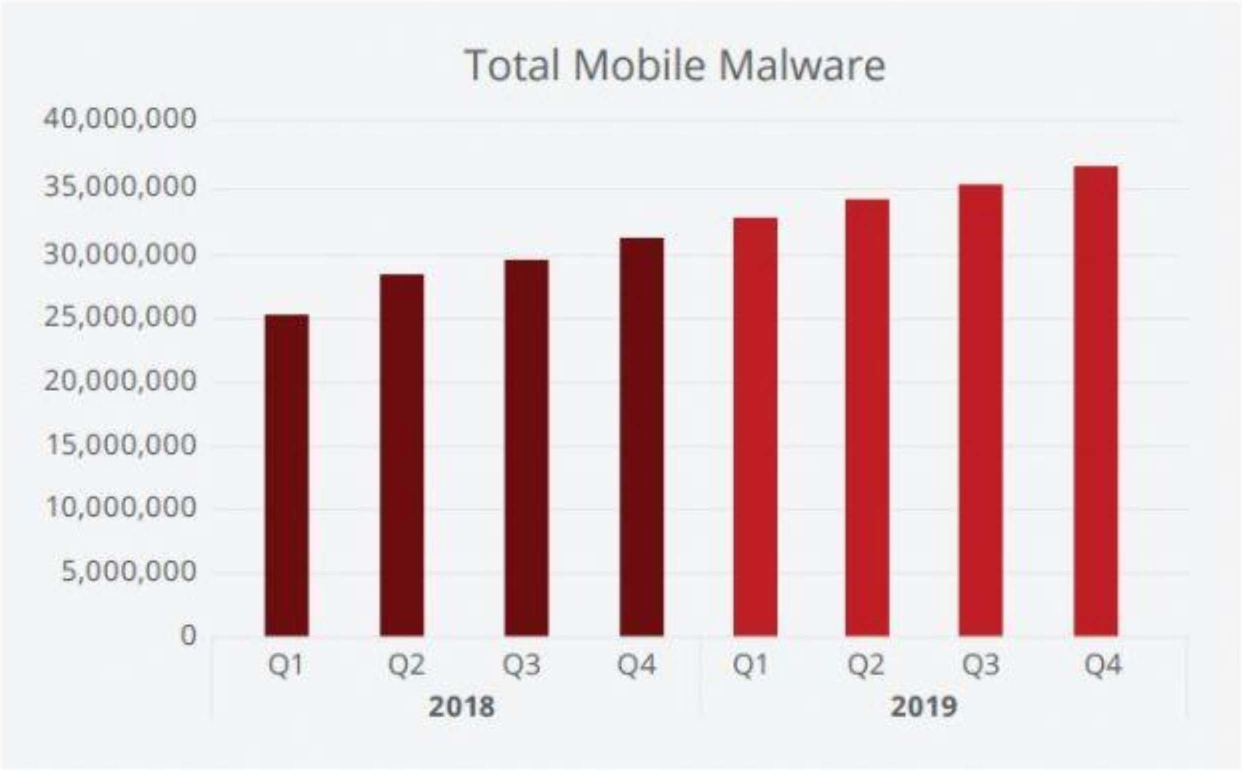
Kang Mun Yeong

Contents

- Abstract
- Related Work
- CNN-based Malware Detection Method
  - Malware Images
  - CNN Model
- Experimental Results
  - Dataset
  - Performance of Family Classification
- Conclusions

Abstract

- McAfee Mobile Threat Report in the first quarter of 2020
  - the total number of mobile malware in the fourth quarter of 2019 reached 35 million
  - 40% increase in the same period last year



Samani, R., and G. Davis. "McAfee Mobile Threat Report Q1." (2019).

Abstract

- The existing malware family classification techniques
  - Use static or dynamic analysis

- Classification of malware families through imaging
  - Time-efficient and uses less computing resources
  - Not require to extract API calls, control-flow graphs, permissions, opcodes, etc. through an analysis of executable files or source codes
  - Not need to consider whether detection bypass techniques such as packing or obfuscation have been applied

- This paper shows the effect of family classification through imaging using two images
  - **Classes.dex file**
  - the other from **a data section inside the Classes.dex file**

Related Work

- Yi-min and Tie-ming, “Android malware family classification method based on the image of bytecode” [2]
  - classifying images using a random forest algorithm after generating images from Dex files.

- Seok and Kim, “Visualize Malware Classification Based-on Convolutional Neural Network” [3]
  - generated an image by converting each byte of the malicious code binary file into 8-bit grayscale pixels
  - classified the malicious code family by applying CNN to the generated image

- Tang and Wang “ConvProtoNet: Deep Prototype Induction towards Better Class Representation for Few-Shot Malware Classification” [4]
  - proposed a new neural network structure called ConvProtoNet to solve the inaccuracy of classification caused by overfitting that occurs when the number of samples is small

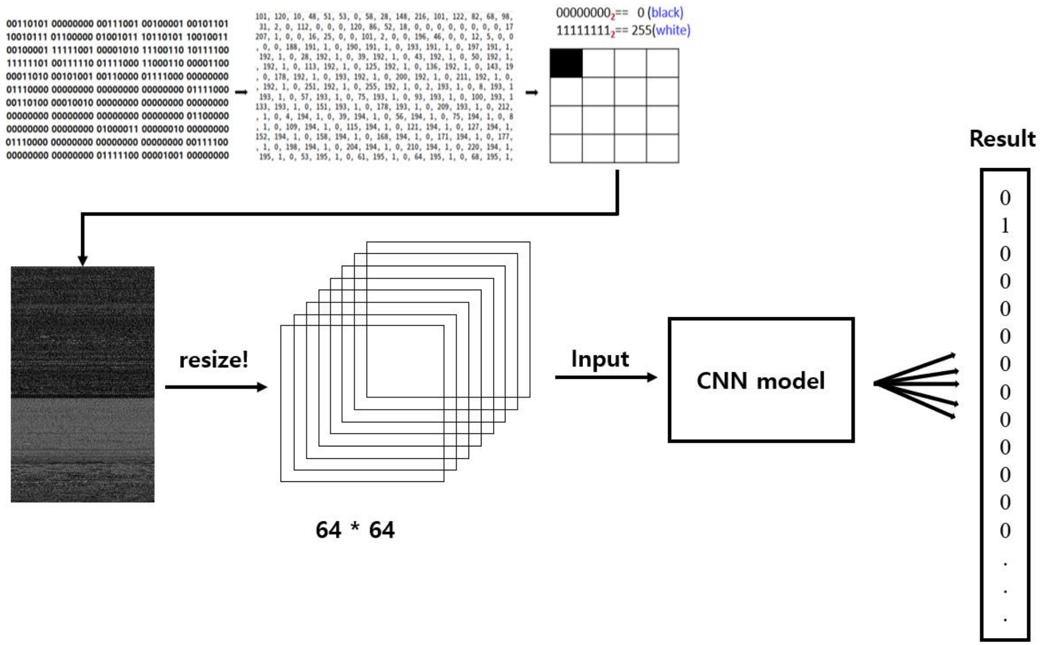
Related Work

- Arp et al “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket” [5]
  - obtained feature information such as permission and API information using static analysis techniques
  - detected malicious apps using machine learning, and classified families

- Jung et al “Android malware detection using convolutional neural networks and data section images”
  - proposed an Android malware detection technique that was based on converting the entire DEX file of an app and the data section of the DEX into gray-scale image
  - classifying the app as benign or malicious using CNN algorithm

CNN-based Malware Detection Method

- The procedure of the proposed method



```
00110101 00000000 00111001 00100001 00101101
10010111 01100000 01001011 10110101 10010011
00100001 11111001 00001010 11100110 10111100
11111101 00111110 01111000 11000110 00001100
00011010 00101001 00110000 01111000 00000000
01110000 00000000 00000000 00000000 01111000
00110100 00010010 00000000 00000000 00000000
00000000 00000000 00000000 00000000 01100000
00000000 00000000 01000011 00000010 00000000
01110000 00000000 00000000 00000000 00111100
00000000 00000000 01111100 00001001 00000000
```

```
101, 120, 10, 48, 51, 53, 0, 58, 28, 148, 216, 101, 122, 82, 68, 98,
 31, 2, 0, 112, 0, 0, 0, 120, 86, 52, 18, 0, 0, 0, 0, 0, 0, 0, 0, 17
207, 1, 0, 0, 16, 25, 0, 0, 101, 2, 0, 0, 196, 46, 0, 0, 12, 5, 0, 0
, 0, 0, 188, 191, 1, 0, 190, 191, 1, 0, 193, 191, 1, 0, 197, 191, 1,
 192, 1, 0, 28, 192, 1, 0, 39, 192, 1, 0, 43, 192, 1, 0, 50, 192, 1,
, 192, 1, 0, 113, 192, 1, 0, 125, 192, 1, 0, 136, 192, 1, 0, 143, 19
, 0, 178, 192, 1, 0, 193, 192, 1, 0, 200, 192, 1, 0, 211, 192, 1, 0,
, 192, 1, 0, 251, 192, 1, 0, 255, 192, 1, 0, 2, 193, 1, 0, 8, 193, 1
 193, 1, 0, 57, 193, 1, 0, 75, 193, 1, 0, 93, 193, 1, 0, 100, 193, 1
133, 193, 1, 0, 151, 193, 1, 0, 178, 193, 1, 0, 209, 193, 1, 0, 212,
, 1, 0, 4, 194, 1, 0, 39, 194, 1, 0, 56, 194, 1, 0, 75, 194, 1, 0, 8
, 1, 0, 109, 194, 1, 0, 115, 194, 1, 0, 121, 194, 1, 0, 127, 194, 1,
152, 194, 1, 0, 158, 194, 1, 0, 168, 194, 1, 0, 171, 194, 1, 0, 177,
, 1, 0, 198, 194, 1, 0, 204, 194, 1, 0, 210, 194, 1, 0, 220, 194, 1,
 195, 1, 0, 53, 195, 1, 0, 61, 195, 1, 0, 64, 195, 1, 0, 68, 195, 1,
```

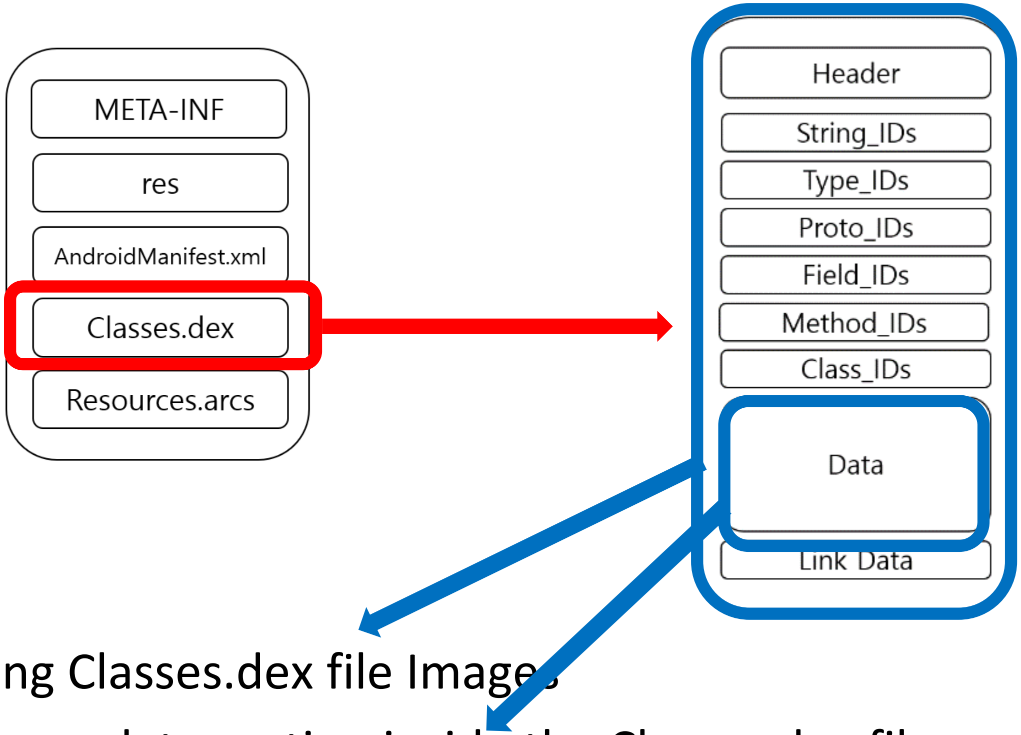
00000000_2== 0 (black)
11111111_2== 255(white)

  - Creating Malware Images
    - Using Classes.dex file Images
    - Using a data section inside the Classes.dex file

  - Classification Using CNN Model

Malware Images

- Dex file Sturcture
- DEX file consists of Header, String_IDs, Type_IDs, Proto_IDs, Field_IDs, Method_IDs, Class_Defs, Data (data section), and Link_Data

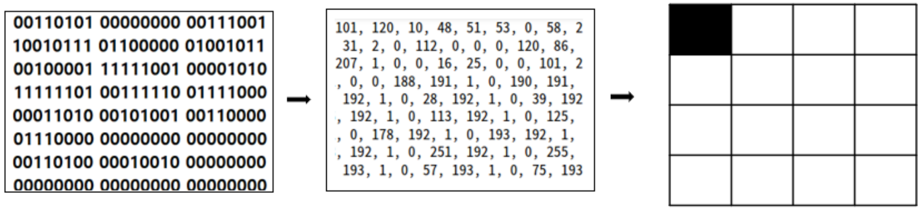


- Using Classes.dex file Images
- Using a data section inside the Classes.dex file

Image Creation

- The files are read as a binary number as 8-bit intervals and interpreted as an unsigned decimal number

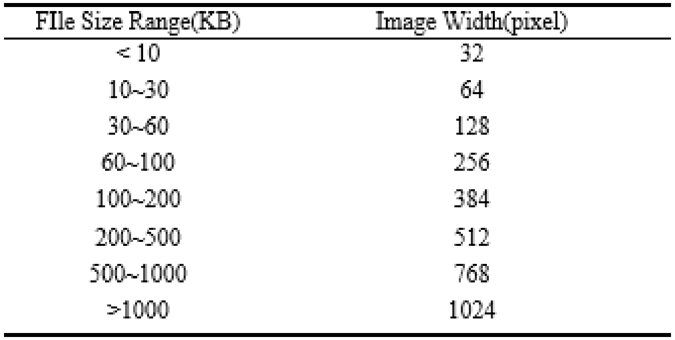
00000000_2== 0 (black)
11111111_2== 255(white)



```
00110101 00000000 00111001
10010111 01100000 01001011
00100001 11111001 00001010
11111101 00111110 01111000
00011010 00101001 00110000
01110000 00000000 00000000
00110100 00010010 00000000
00000000 00000000 00000000
```

```
101, 120, 10, 48, 51, 53, 0, 58, 2
 31, 2, 0, 112, 0, 0, 0, 120, 86,
207, 1, 0, 0, 16, 25, 0, 0, 101, 2
, 0, 0, 188, 191, 1, 0, 190, 191,
 192, 1, 0, 28, 192, 1, 0, 39, 192
, 192, 1, 0, 113, 192, 1, 0, 125,
, 0, 178, 192, 1, 0, 193, 192, 1,
, 192, 1, 0, 251, 192, 1, 0, 255,
 193, 1, 0, 57, 193, 1, 0, 75, 193
```

- Numbers converted to decimal are numbers from 0 to 255
  - each pixel is associated with a number representing the brightness.
  - 0 represents black and 255 represents white.
- The image is represented by a two-dimensional array.
  - the relationship between the file size and the size of the generated image file.

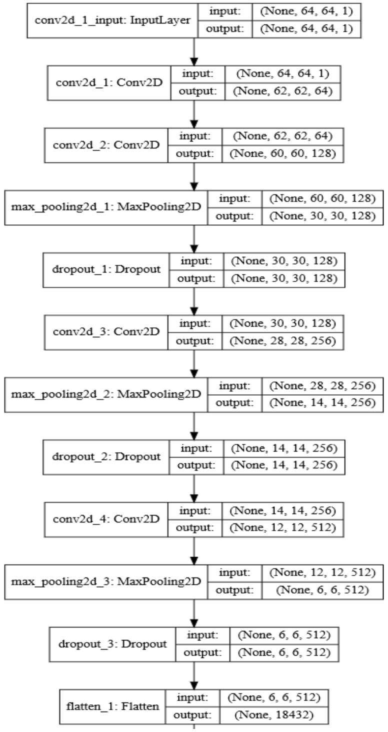


FIle Size Range(KB)	Image Width(pixel)
< 10	32
10~30	64
30~60	128
60~100	256
100~200	384
200~500	512
500~1000	768
>1000	1024

CNN Model

- Use LeNet (or LeNet-5) which is one of the famous CNN architectures
  - The convolution layer can reduce number of learnable parameters and introduce translation invariance
  - Max pooling outputs the maximum value in each patch, and discards all the other values

- CNN's input data, have been reduced to a size of 64*64*1.


conv2d_1_input: InputLayer | input: | (None, 64, 64, 1) | output: | (None, 64, 64, 1)
conv2d_1: Conv2D | input: | (None, 64, 64, 1) | output: | (None, 62, 62, 64)
conv2d_2: Conv2D | input: | (None, 62, 62, 64) | output: | (None, 60, 60, 128)
max_pooling2d_1: MaxPooling2D | input: | (None, 60, 60, 128) | output: | (None, 30, 30, 128)
dropout_1: Dropout | input: | (None, 30, 30, 128) | output: | (None, 30, 30, 128)
conv2d_3: Conv2D | input: | (None, 30, 30, 128) | output: | (None, 28, 28, 256)
max_pooling2d_2: MaxPooling2D | input: | (None, 28, 28, 256) | output: | (None, 14, 14, 256)
dropout_2: Dropout | input: | (None, 14, 14, 256) | output: | (None, 14, 14, 256)
conv2d_4: Conv2D | input: | (None, 14, 14, 256) | output: | (None, 12, 12, 512)
max_pooling2d_3: MaxPooling2D | input: | (None, 12, 12, 512) | output: | (None, 6, 6, 512)
dropout_3: Dropout | input: | (None, 6, 6, 512) | output: | (None, 6, 6, 512)
flatten_1: Flatten | input: | (None, 6, 6, 512) | output: | (None, 18432)

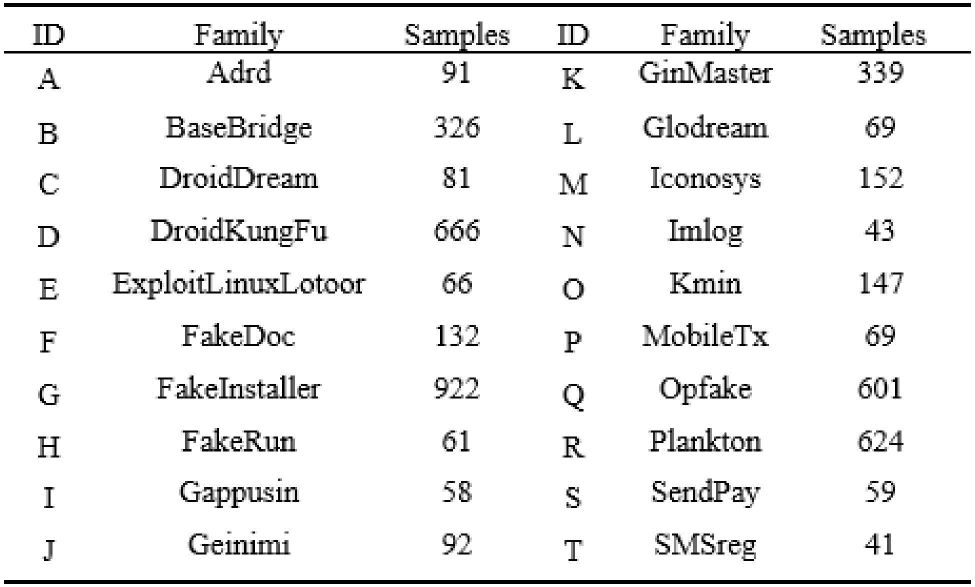
Experimental results

- Performance comparison with other methods
    - Compare the proposed method with other techniques using the same data set
    - This shows that the proposed technique is effective for malware classification

- Performance comparison when using the image of the entire DEX file and the image of the data section only:
    - compare the performance when using the image of the entire Dex file and the image of the data section only and explore whether to reduce the overhead of classification

Dataset

- Use the dataset collected by the Drebin Project. The dataset consists of 5,560 malicious apps consisting of 179 different families
  - Among them, 5,528 apps remain, excluding 28 files without Dex files and 4 files with APK file structure errors.
  - Experiment with apps belonging to the top 20 families with a large number of family members
- The ratio between the training set and the test set is 8:2



ID	Family	Samples	ID	Family	Samples
A	Adrd	91	K	GinMaster	339
B	BaseBridge	326	L	Glodream	69
C	DroidDream	81	M	Iconosys	152
D	DroidKungFu	666	N	Imlog	43
E	ExploitLinuxLotoor	66	O	Kmin	147
F	FakeDoc	132	P	MobileTx	69
G	FakeInstaller	922	Q	Opfake	601
H	FakeRun	61	R	Plankton	624
I	Gappusin	58	S	SendPay	59
J	Geinimi	92	T	SMSreg	41

Dataset

- Average size of Image Files for each Family
- Average 18% difference in size for the entire family

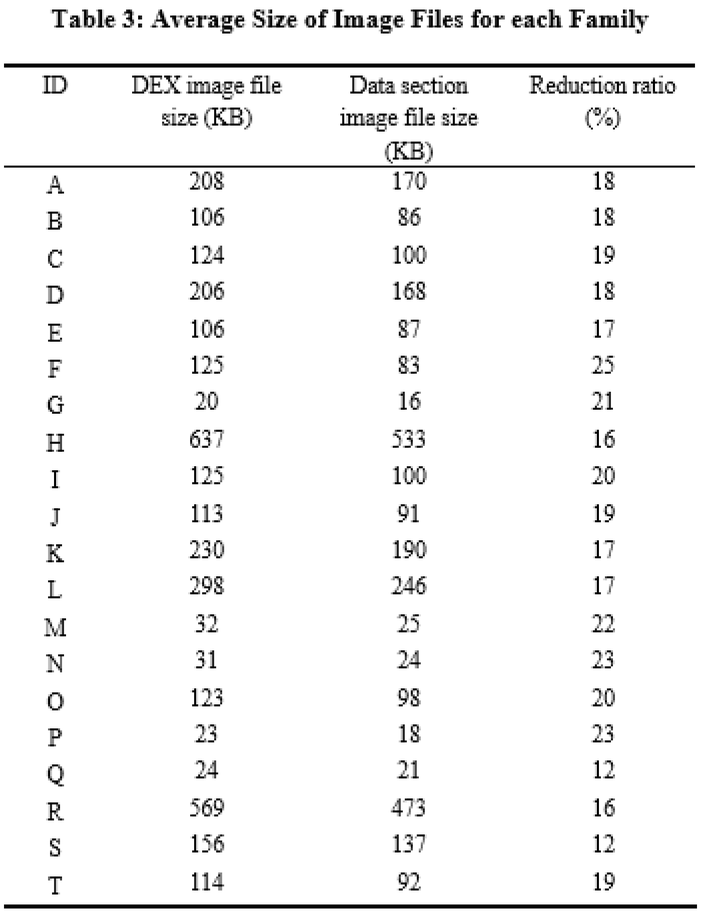


Table 3: Average Size of Image Files for each Family

ID	DEX image file size (KB)	Data section image file size (KB)	Reduction ratio (%)
A	208	170	18
B	106	86	18
C	124	100	19
D	206	168	18
E	106	87	17
F	125	83	25
G	20	16	21
H	637	533	16
I	125	100	20
J	113	91	19
K	230	190	17
L	298	246	17
M	32	25	22
N	31	24	23
O	123	98	20
P	23	18	23
Q	24	21	12
R	569	473	16
S	156	137	12
T	114	92	19

Performance of Family Classification

- Performance comparison with other methods
  - DREBIN shows 93% accuracy
    - Analyze the permission requested by the app through static analysis
    - Extract API usage information, and classifies the family of malicious apps through machine learning.
  - Proposed method is effective in terms of detection time and resource usage
  - Only goes through the process of converting DEX files or Data Sections into images

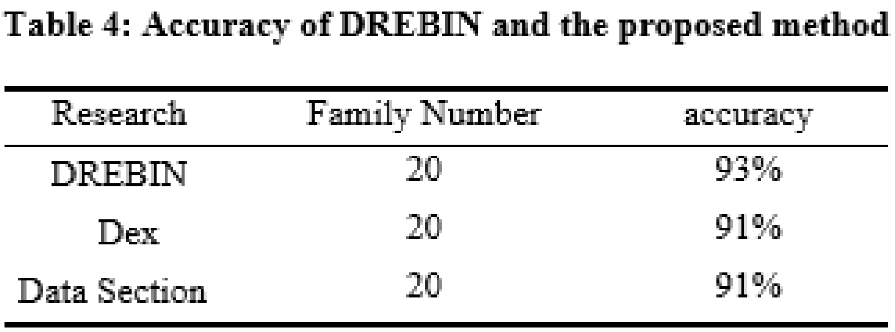


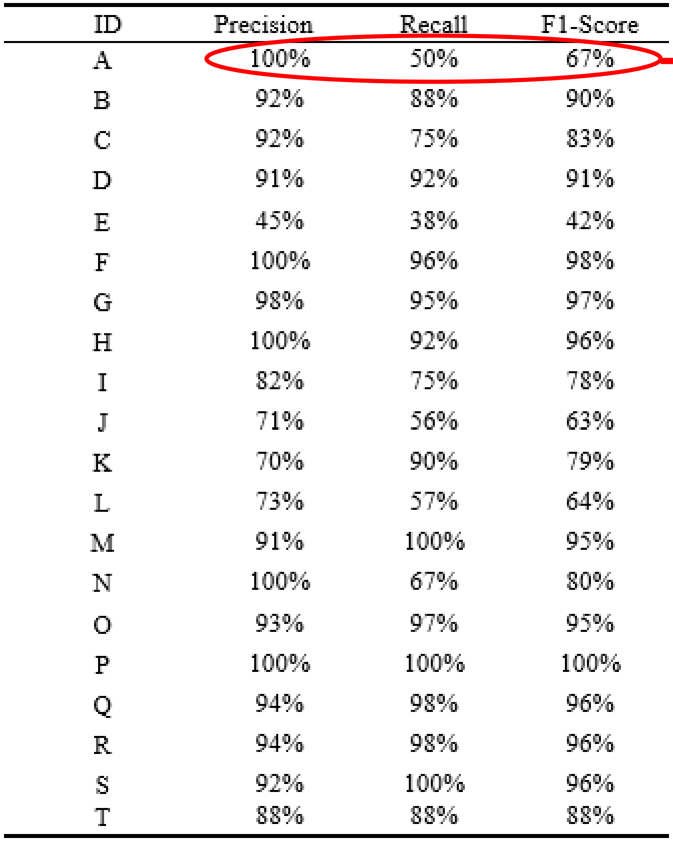
Table 4: Accuracy of DREBIN and the proposed method

Research	Family Number	accuracy
DREBIN	20	93%
Dex	20	91%
Data Section	20	91%

Performance of Family Classification

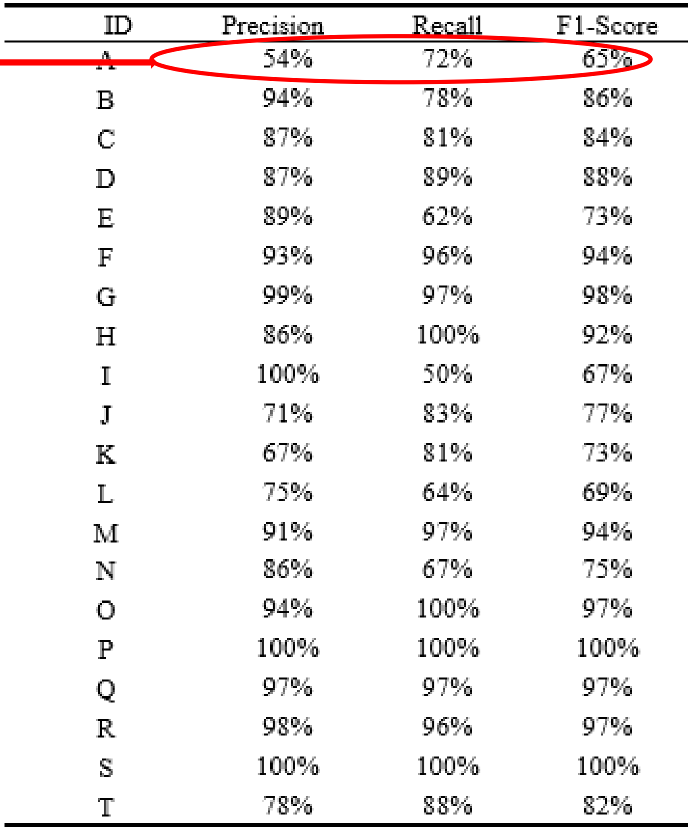
- Performance comparison when using the DEX file and the data section image

- Dex Image



ID	Precision	Recall	F1-Score
A	100%	50%	67%
B	92%	88%	90%
C	92%	75%	83%
D	91%	92%	91%
E	45%	38%	42%
F	100%	96%	98%
G	98%	95%	97%
H	100%	92%	96%
I	82%	75%	78%
J	71%	56%	63%
K	70%	90%	79%
L	73%	57%	64%
M	91%	100%	95%
N	100%	67%	80%
O	93%	97%	95%
P	100%	100%	100%
Q	94%	98%	96%
R	94%	98%	96%
S	92%	100%	96%
T	88%	88%	88%

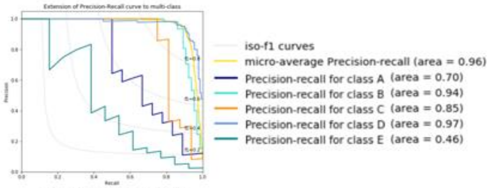
- Data Section Image

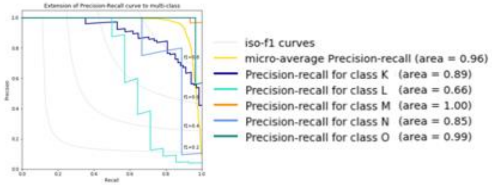


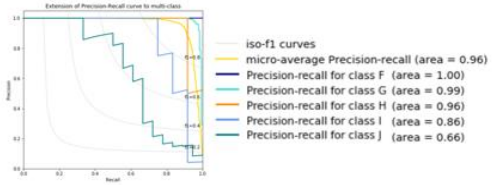
ID	Precision	Recall	F1-Score
A	54%	72%	65%
B	94%	78%	86%
C	87%	81%	84%
D	87%	89%	88%
E	89%	62%	73%
F	93%	96%	94%
G	99%	97%	98%
H	86%	100%	92%
I	100%	50%	67%
J	71%	83%	77%
K	67%	81%	73%
L	75%	64%	69%
M	91%	97%	94%
N	86%	67%	75%
O	94%	100%	97%
P	100%	100%	100%
Q	97%	97%	97%
R	98%	96%	97%
S	100%	100%	100%
T	78%	88%	82%

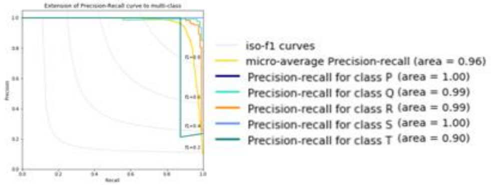
Performance of Family Classification

- Precision-recall curve for each family of the results with the Dex file image and Data Section image

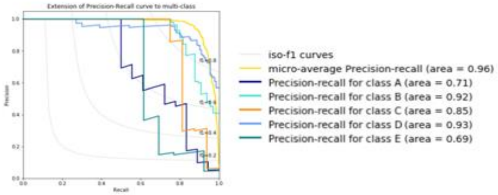


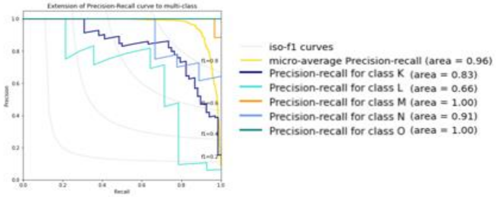


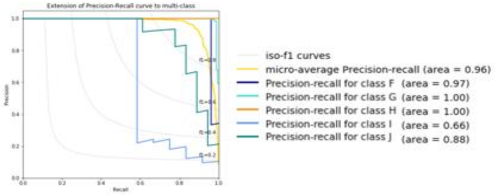


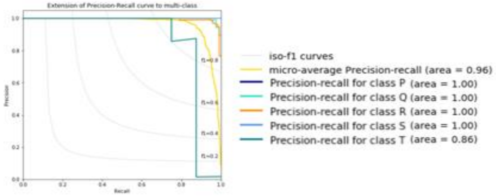


- Dex Image





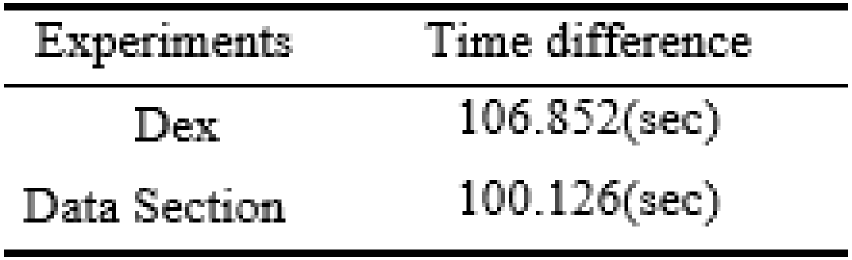




- Data Section Image

Performance of Family Classification

- Comparing the training time



Experiments	Time difference
Dex	106.852(sec)
Data Section	100.126(sec)

Concluision

- Using Image processing Technique
  - In both experiments using images, the methods classifies malware family with an average of 91% accuracy
  - this technique has been shown to be effective in saving time and resource use
- Using Data Section Image
  - accuracy is almost the same
  - training time is also shortened
  - However, some families showed poorer performance when using the Data Section image
- Research is needed to improve performance by combining API information and permission information as well as images

Thank you