Winter 2017

Malware Detection using the Index of Coincidence

Bhavna Gurnani
San Jose State University

Malware Detection using the Index of Coincidence

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Bhavna Gurnani

Dec 2016

The Designated Project Committee Approves the Project Titled

Malware Detection using the Index of Coincidence

by

Bhavna Gurnani

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Dec 2016

Dr. Mark Stamp	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Mr. Fabio Di Troia	Department of Computer Science

ABSTRACT

Malware Detection using the Index of Coincidence

by Bhavna Gurnani

In this research, we apply the Index of Coincidence (IC) to problems in malware analysis. The IC, which is often used in cryptanalysis of classic ciphers, is a technique for measuring the repeat rate in a string of symbols. A score based on the IC is applied to a variety of challenging malware families. We find that this relatively simple IC score performs surprisingly well, with superior results in comparison to various machine learning based scores, at least in some cases.

ACKNOWLEDGMENTS

I am grateful to Dr. Stamp for providing me with the opportunity to work on this project and guiding me. I would like to thank my committee members for reviewing my work. Also, special thanks to Fabio Di Troia for helping me throughout the project. I would also like to thank my brother for constantly motivating me during the project.

TABLE OF CONTENTS

CHAPTER

APPENDIX

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Introduction to Malware

Malware [2] is an umbrella term for various kinds of malicious software. Such programs are designed to harm computers in several ways including stealing personal information, installing illegitimate software, reformatting hard drives, deleting files and so on. Only six years after the launch of the first personal computer in 1975, the world was presented with its first computer virus [13]. Ever since, malware has seemed unstoppable. The rate with which malicious code and other unwanted programs are released is increasing in comparison to the release rate of legitimate software applications. In this paper, the terms virus and malware are used interchangeably.

1.2 Types of Malware

There has been an arms race between malware writers and anti-virus software publishers. The most commonly and widely used malware detection technique is signature detection [2]. In this technique, pattern matching is used to scan for particular byte patterns that are found in malware. If a matching pattern is found, the file is categorized as malware. This is a very simple and often effective technique, but malware writers have developed various techniques aimed at defeating signature detection.

Encrypting the body of a virus can effectively defeat signature detection. However, the decryption routine is not encrypted, and can be susceptible to signature scanning. As a result, malware authors created the polymorphic [14] and the metamorphic virus [6].

A polymorphic virus includes an encrypted body and decryption code. Additionally, polymorphic malware changes the decryption routine with each new infection, making straightforward signature detection infeasible. However, anti-virus software can use emulation to detect polymorphic malware. At some point the malware will decrypt itself and at that point it will be vulnerable to signature detection.

Metamorphic malware are sometimes said to be “body polymorphic” [6], as they morph the entire body of the malware with each new infection. This changes the internal structure and, if sufficiently thorough, it will defeat signature scanning. Encryption is not necessary, and hence is not used, in metamorphic malware. Metamorphic code can employ a wide variety of code morphing strategies [10].

1.3 Techniques to detect malware

Proposed malware detection mechanisms rely on static analysis or dynamic analysis or some combination thereof [1, 8, 12, 15]. In static analysis, the necessary features are executed without executing the code. Examples of such features include opcode sequences, entropy, and so on. In contrast, dynamic analysis extracts the features by executing (or emulating) code. Dynamic features can be used to determine aspects of the actual behavior of the code.

1.4 Technique used

This thesis is based on static analysis, including the opcode generation feature. Different malware families are identified using a popular signature detection technique known as the Index of Coincidence (IC).

The contents of this report are presented as follows. Chapter 2, briefly describes the previous work and explains in detail about the Index of Coincidence and Chapter 3 its implementation. Chapter 4 includes several results after implementing IC and Chapter 5 presents some observations based on the results. Chapter 6 concludes the report.

CHAPTER 2

Background

2.1 Previous work and research

Cryptography is the technique of translating a piece of plain text to cipher text. Classical ciphers are some of the simplest ciphers existing today. Most common forms of classical ciphers are substitution ciphers and transposition ciphers. Substitution ciphers are then classified into monoalphabetic and polyalphabetic substitution ciphers. Monoalphabetic ciphers are vulnerable to attacks and so are easily identified [9]. Alberti used this vulnerability of monoalphabetic ciphers to create polyalphabetic ciphers [11], which were difficult to break for centuries.

Previous work includes the implementation of several techniques like the Simple Substitution Distance and the Vigenère Cipher Score for the cryptanalysis of classical ciphers for malware detection. Simple Substitution Distance (SSD), uses the hill climbing [7] mathematical technique to first find or guess a possible solution and then alters the solution to find a more efficient solution. A score is generated to compare the previous and current solutions. If the current solution is better, then an incremental change is made to the solution. Otherwise, the solution remains the same.

The Vigenère Cipher is a polyalphabetic cipher that is formed by interweaving a series of simple substitution ciphers [4]. To break the Vigenère cipher one needs to first find the key length and then divide the cipher text into that many columns. One can then derive the original key to decrypt the Vigenère cipher using frequency analysis for each column, Index of Coincidence, also known as coincidence counting, will be implemented in this project. This technique calculates the frequency of opcodes to measure the score.

2.2 Index of Coincidence

Index of coincidence, usually used for cryptanalysis of classic ciphers, is a technique for measuring the repeat rate or frequency in a piece of text or a string of symbols. Index of Coincidence [5] is defined as the probability of selecting the same letter twice, when randomly selecting two letters from the piece of text under consideration.

2.2.1 Calculation

The Index of Coincidence is calculated in several steps. First, the chance of selecting a letter from the text is calculated. Mathematically, this is the frequency a letter in the text divided by the length of the text. Next, the chance of selecting the same letter without replacing the first drawn letter is calculated. This involves dividing (frequency - 1) by (length -1). By multiplying these two values, the probability of that particular letter is obtained. In order to get the chances of two of a kind from the text, we find the probability of each letter and sum those values. Then by multiplying the sum with a normalizing coefficient, typically 26 for English, we obtain the value of IC.

$$IC = c * ((\frac{n_a}{N} * \frac{n_a - 1}{N - 1}) + (\frac{n_b}{N} * \frac{n_b - 1}{N - 1}) + ... + (\frac{n_z}{N} * \frac{n_z - 1}{N - 1})) \tag{1}$$

where, c is the normalizing coefficient, n_a is the frequency of the letter ‘a’ in the text and N is length of the text. IC can also be calculated as mentioned in Equation 2.

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{\frac{N(N-1)}{c}} \tag{2}$$

where, N is the length of the text and n_1 through n_c are the frequencies of the first c letters. Summation of n_i should be equal to N.

2.2.2 Application

IC is useful in the analysis of natural-language plaintext as well as ciphertext [5]. This technique is helpful even when the plaintext is not available and only the ciphertext is present. During such times, coincidences in the ciphertext are caused since they are present in the plain text as well. Coinciding counting is helpful in the cryptanalysis of the Vigenère cipher. More details on the calculation of IC are covered in the next chapter.

CHAPTER 3

Implementation

3.1 Choices Made in Calculation

Calculating IC mainly depends on different choices. From the last chapter, there are three major choices involved in IC calculation.

- Value of the normalizing coefficient, c.
- Training family
- Testing family

The normalizing coefficient, as seen in the previous chapter, is the value or constant with which the rest of the IC term is multiplied. For the English language, c is equal to the number of distinct letters, which is 26. But, as the input here is a set of opcodes, the letters will be replaced by the opcodes. So, different values of c should be considered while calculating the value of IC. While calculating IC, the distinct opcodes present in a family are obtained, so that the top c opcodes can be considered and the remaining opcodes are then combined into one category of 'others'. This process happens only for the training malware family. Whereas, once the top c opcodes are selected for the training family, only those opcodes are considered for the testing family. The remaining opcodes in the testing family are combined into one category of 'others'.

3.2 Process to calculate IC

The first step involved in calculating IC, is to get a sequence of opcodes from the .asm files. This sequence of opcodes is considered to be the cipher text for the generation of IC.

To train a family, unique opcodes and their frequencies are generated. The c most frequent opcodes are considered. The remaining opcodes and their frequencies are combined into one 'other' opcode. These top c opcodes and the 'other' opcode are then used to generate the IC of the training family. The opcodes considered during this process are used for testing different files and families. Let us call these opcodes as 'considered_opcodes'.

Testing files of the training family includes calculating the frequency of distinct opcodes in the file. Opcodes present in 'considered_opcodes' are considered and the remaining opcodes and their frequency is summed into 'other' opcodes. Using these opcodes, the IC for a family is generated. This process is repeated for each file in the family.

For the final score calculation, the IC for both families and files are considered. Let us call the score of family as x and score of file as y. Then, the final score of the file will be

$$|z| = |y - x| \tag{3}$$

where, $|z|$ is the absolute value of z, which is the final score of a file. The closer the value is to 0, the better it matches the expected value for the family.

To test the testing family, all the opcodes from all the files in that family are combined. Then, unique opcodes and their frequencies are generated. Using only those opcodes which are present in 'considered_opcodes' the sum of the frequencies of the remaining opcodes in 'other' opcodes is calculated. Using the formula, the IC for that family is calculated and the process is repeated for the test family. Once the final scores for all the files in both families are obtained, ROC curves and AUC values are generated [3] as described in the next chapter.

CHAPTER 4

Results

Several results were obtained based on different malware families like NGVCK, harebot, zbot, zeroaccess, cleaman, cridex. The datasets considered in this project are shown in Table 1.

Family	Number of files	Distinct opcodes
NGVCK	200	79
Harebot	54	207
Zeroaccess	230	412
Zbot	242	320
Security Shield	59	177
Winwebsec	161	334
Smart HDD	69	85
Cridex	75	254
Cygwin	40	196
Benign	40	117

Table 1: Dataset

To generate results and to decide the value of the normalizing coefficient, distinct opcodes of each malware family were obtained. Hence, the count of the distinct opcodes was also included in Table 1.

Considering the NGVCK and the Benign datasets, and using different values of c, some IC scores were computed and plotted as shown in Figure 1 and Figure 2.

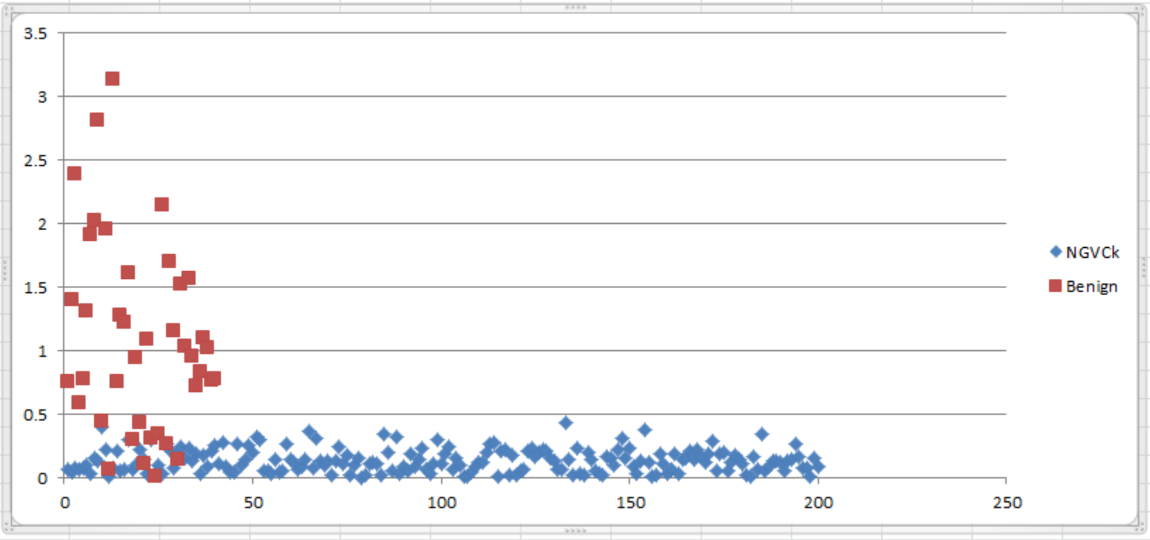

Figure 1: NGVCK-Benign scatter plot, *c* is 26

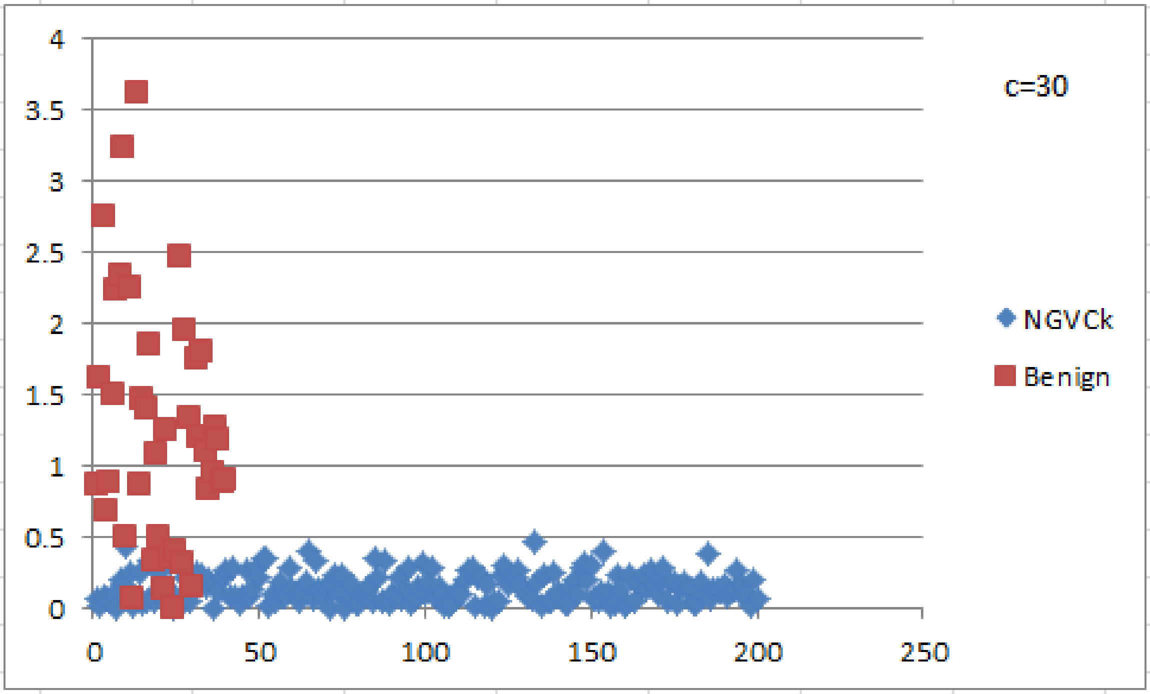

Figure 2: NGVCK-Benign scatter plot, *c* is 30

Observing the graph, there is no clear distinction between the two families. Hence, for more clarity, AUC values and ROC curves were generated. Results were classified into different categories like variations in the values of the normalizing coefficient or different combinations of training and testing datasets.

4.1 With variation of normalizing coefficient

In this section, the results were calculated with different values of the normalizing coefficient. To generate results NGVCK is used as the training dataset and Benign is used as the testing dataset. Unique opcodes in the training dataset were found to determine the range for the value of c. There were 79 unique opcodes in NGVCK, so the values of c cannot exceed 79. AUC were generated with different values of c and plotted as shown in Figure 3 and Figure 4.

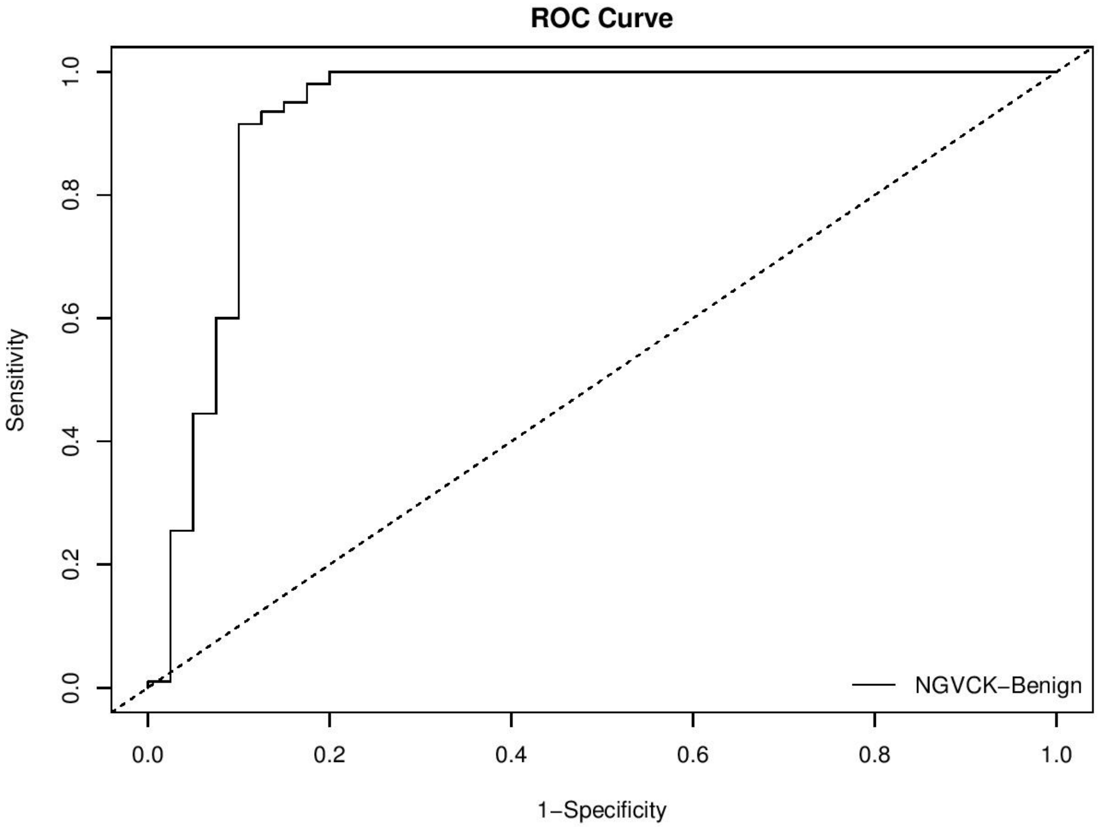

Figure 3: ROC curve for c as 26

c	Training dataset	Testing dataset	AUC
26	NGVCK	Benign	0.92725

Table 2: Parameters for c as 26

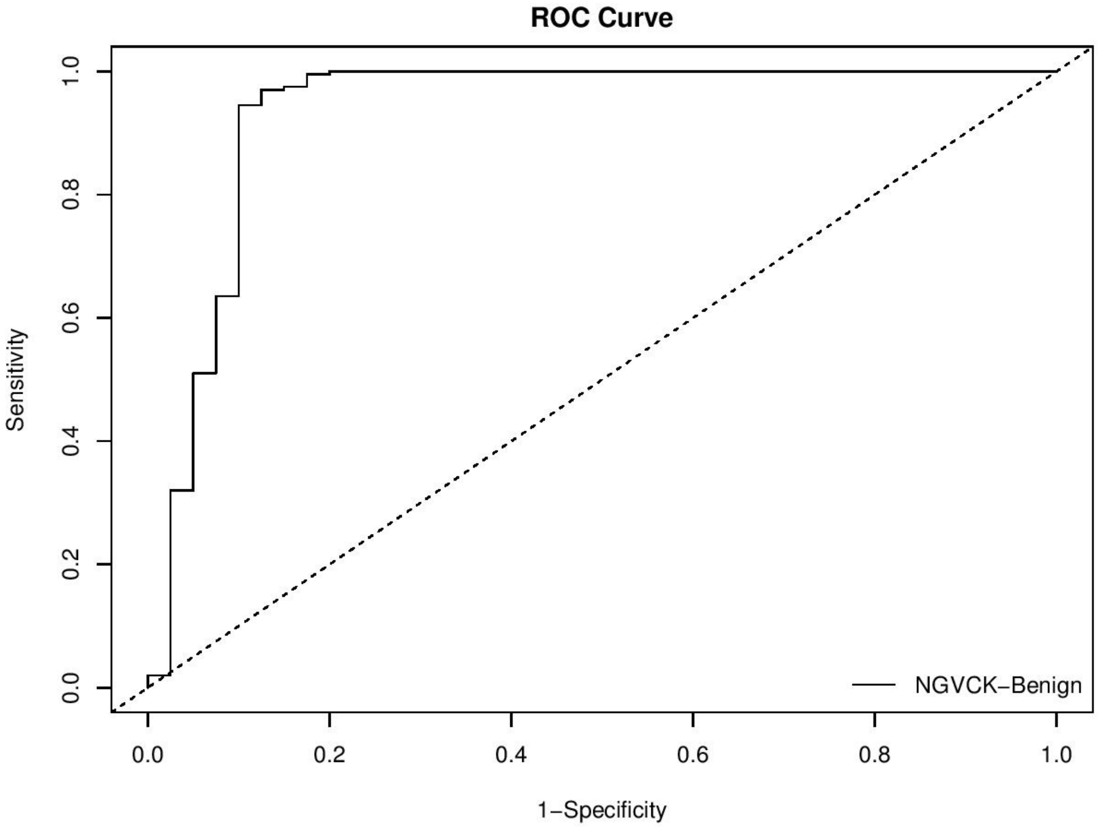

Figure 4: ROC curve for c as 79

c	Training dataset	Testing dataset	AUC
79	NGVCK	Benign	0.93425

Table 3: Parameters for c as 79

AUC generated with different values of c are given in Table 4.

c	Training dataset	Testing dataset	AUC
79	NGVCK	Benign	0.93425
63	NGVCK	Benign	0.933125
50	NGVCK	Benign	0.93325
45	NGVCK	Benign	0.933375
34	NGVCK	Benign	0.931875
12	NGVCK	Benign	0.7775
6	NGVCK	Benign	0.7825

Table 4: Results with different normalizing coefficients

In order to obtain a pattern where the result changes, graphs were plotted as shown in Figure 5 and Figure 6.

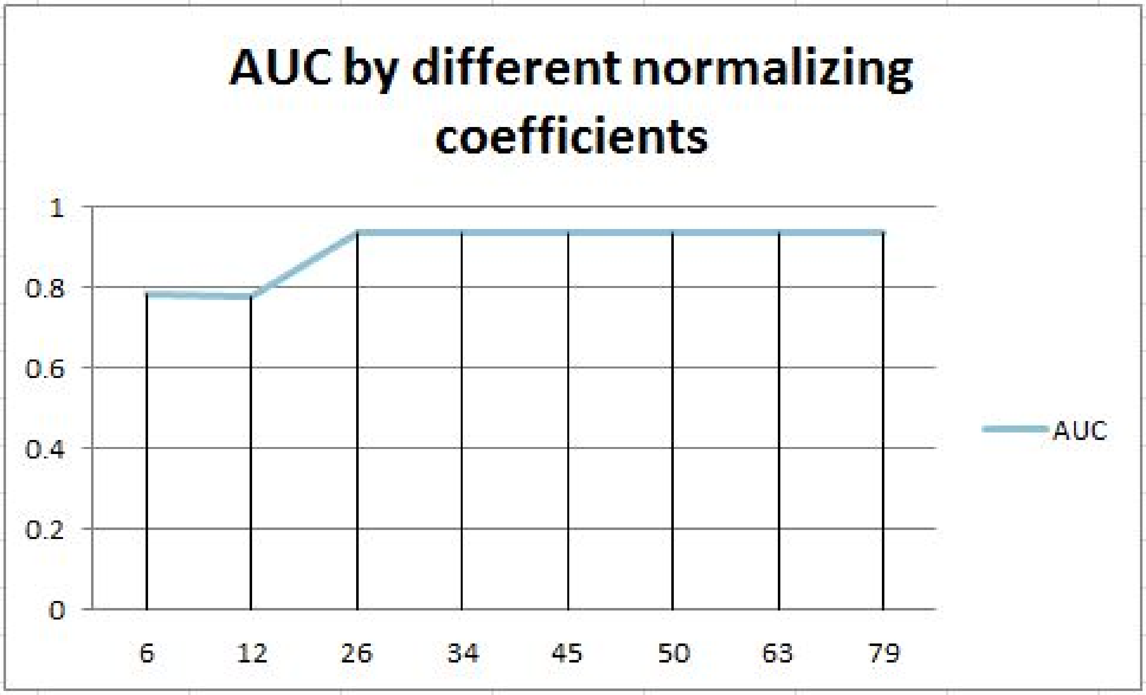

Figure 5: AUC with different normalizing coefficients: line graph

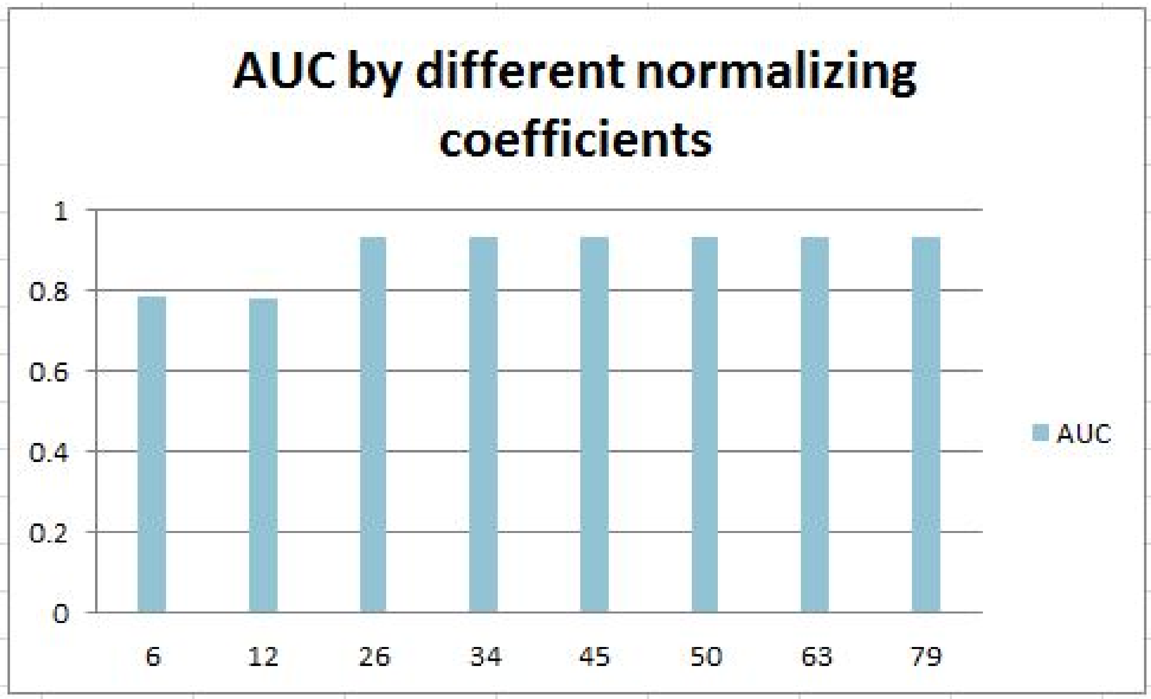

Figure 6: AUC with different normalizing coefficients: bar graph

For more observations, the results were computed with different values of the normalizing coefficient for several training datasets, using benign as the test dataset. The values computed for this scenario are given in Table 5 and plotted in Figure 7.

c	Training dataset	Testing dataset	AUC
117	Harebot	Benign	0.7124521
117	Zeroaccess	Benign	0.7124521
117	Zbot	Benign	0.455735
79	NGVCK	Benign	0.93425
79	Harebot	Benign	0.3
79	Zeroaccess	Benign	0.7127011
79	Zbot	Benign	0.456449
63	NGVCK	Benign	0.933125
63	Harebot	Benign	0.3023585
63	Zeroaccess	Benign	0.712318
63	Zbot	Benign	0.456
50	NGVCK	Benign	0.93325
50	Harebot	Benign	0.3037736
50	Zeroaccess	Benign	0.7130651
50	Zbot	Benign	0.4552903
45	NGVCK	Benign	0.933375
45	Harebot	Benign	0.3051887
45	Zeroaccess	Benign	0.7132759
45	Zbot	Benign	0.4566245
34	NGVCK	Benign	0.931875
34	Harebot	Benign	0.309434
34	Zeroaccess	Benign	0.7147318
34	Zbot	Benign	0.4566245
26	NGVCK	Benign	0.92725
26	Harebot	Benign	0.3188679
26	Zeroaccess	Benign	0.7189464
26	Zbot	Benign	0.4560744
12	NGVCK	Benign	0.7775
12	Harebot	Benign	0.4462264
12	Zeroaccess	Benign	0.6612069
12	Zbot	Benign	0.3904846
2	NGVCK	Benign	0.8035
2	Harebot	Benign	0.4820755
2	Zeroaccess	Benign	0.6356897
2	Zbot	Benign	0.03591994

Table 5: Results with different c and training datasets

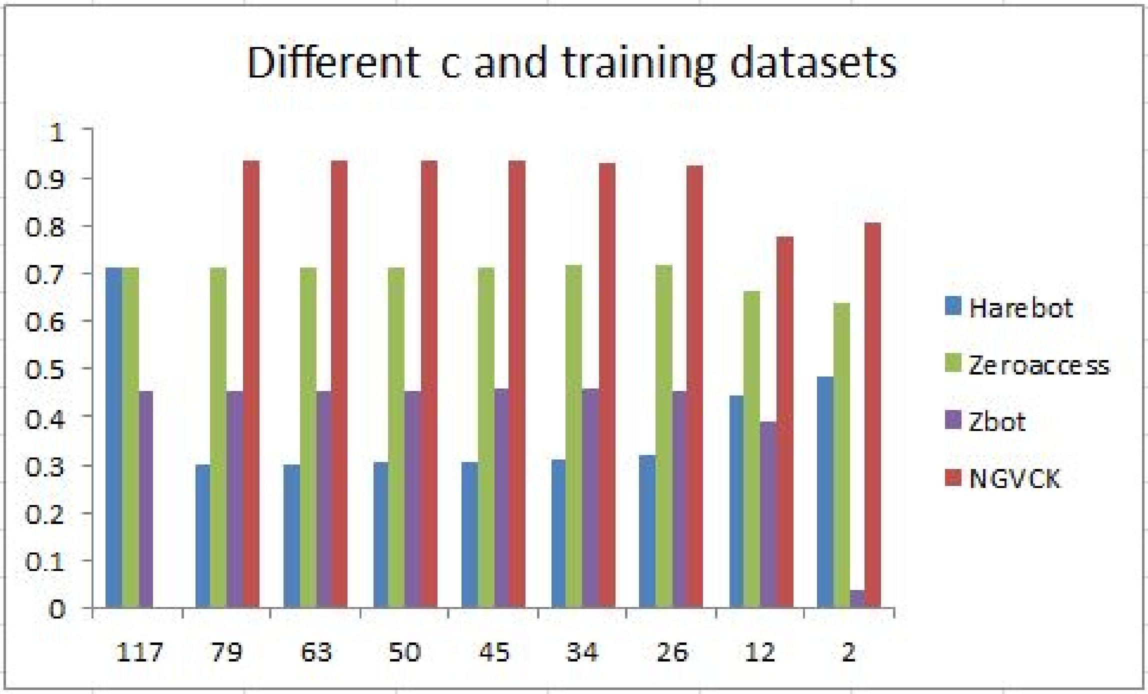

Figure 7: Different *c* and train dataset across benign

It was observed that there is an increase in AUC with the increase in value of the normalizing coefficient. So, one can summarize the above results by saying that the value of *c* is directly proportional to the AUC. But, the values of AUC differ with each training dataset. This indicates that the occurrence of opcodes and their distribution plays an important part in determining the AUC.

4.2 With variation of Train dataset

In this section the values of *c* (26) and test dataset (Benign) are constant. The training dataset changes from NGVCK to Zeroaccess to Smart HDD. IC was calculated with these parameters and ROC curves were generated as shown in Figure 8 and Figure 9.

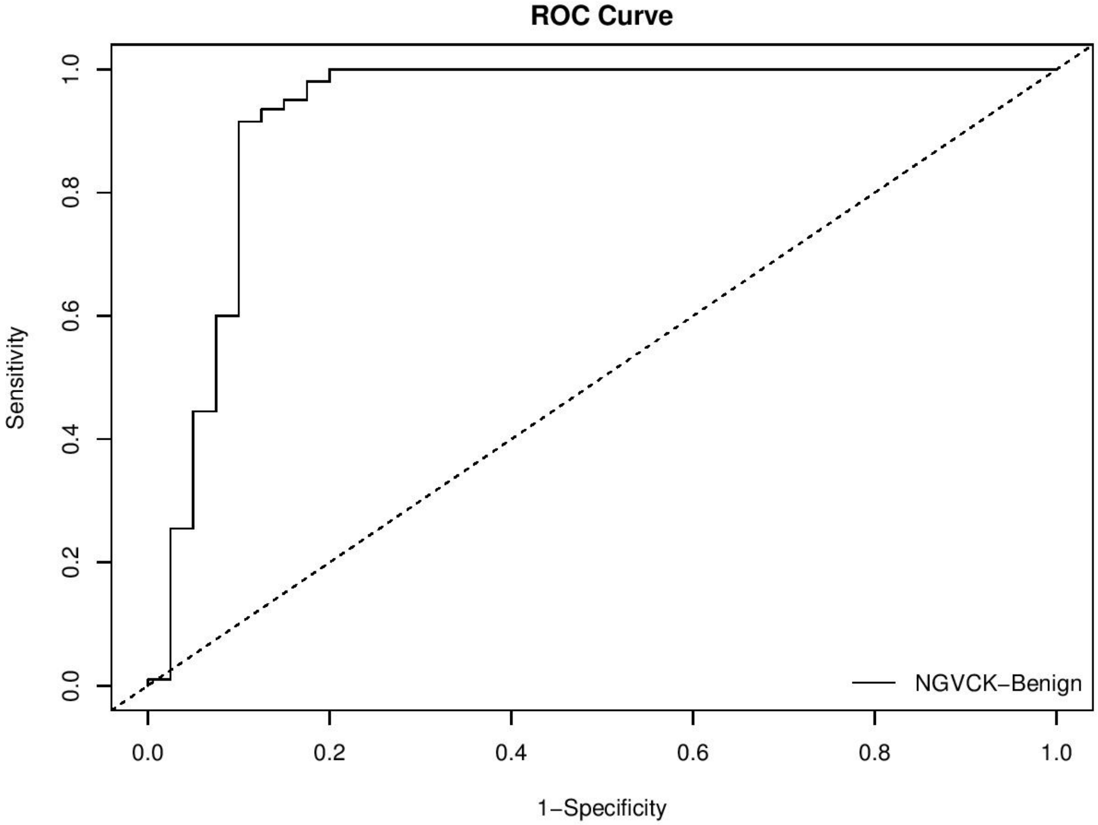

Figure 8: ROC curve for NGVCK as training dataset

c	Training dataset	Testing dataset	AUC
26	NGVCK	Benign	0.92725

Table 6: Parameters for NGVCK as training dataset

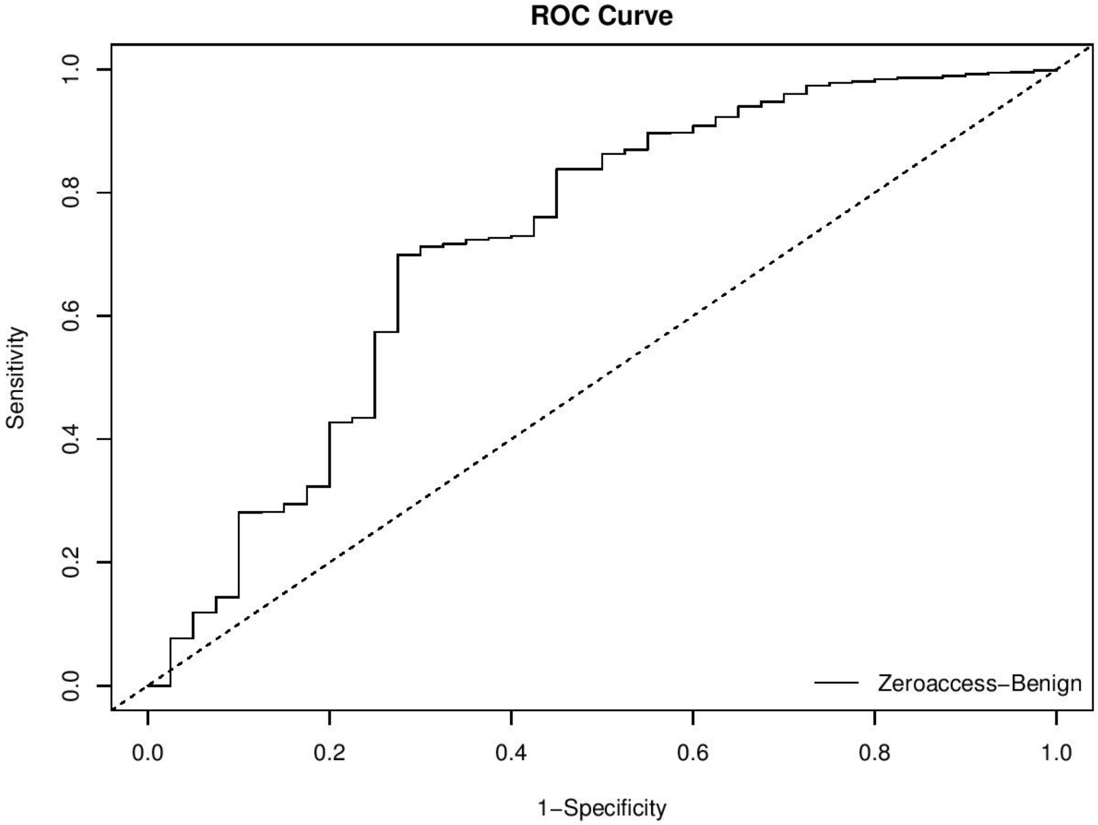

Figure 9: ROC curve for Zeroaccess as training dataset

c	Training dataset	Testing dataset	AUC
26	ZeroAccess	Benign	0.7189

Table 7: Parameters for Zeroaccess as training dataset

Some more results were computed based on the above scenario and are given in Table 8.

c	Training dataset	Testing dataset	AUC
26	NGVCK	Benign	0.92725
26	NGVCK	Cleamen	0.95875
26	NGVCK	Cridex	0.9235135
26	ZeroAccess	Benign	0.7189464
26	Smart HDD	Benign	0.6551471
26	Security Shield	Benign	0.512931
26	Winwebsec	Benign	0.4600287
26	Zbot	Benign	0.4560744
26	Cridex	Benign	0.4351351
26	Cleamen	Benign	0.4132
26	Harebot	Benign	0.3188679

Table 8: Results with different training datasets

These results were generated by changing the training dataset. So, to make the results more visually clear and to find the pattern in the result, bar graph was plotted as shown in Figure 10.

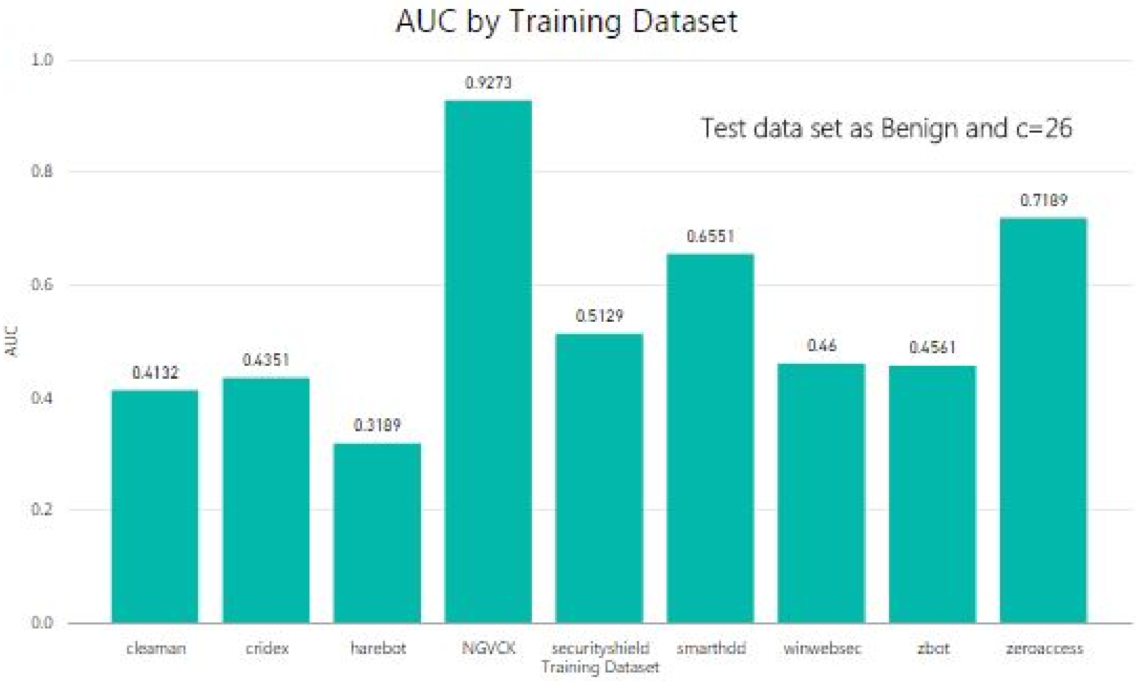

Figure 10: AUC by different training datasets

The above graph shows that the NGVCK training dataset has the highest AUC value of 0.9273, whereas zeroaccess, smart HDD and security shield have AUC values 0.7189, 0.6551 and 0.5129 respectively, all of which are above 0.5. This means that the IC score did a pretty good job with NGVCK as training dataset and a moderately good job with families like zeroaccess, smart hdd and security shield. Other families, when used as the training dataset were not detected accurately with the IC score.

4.3 With variation of Test dataset

Similar to the above two sections, the results in this section were computed by changing the values of the test dataset and the training dataset and the normalizing coefficient were constant. Several families like Cridex and Cleaman were considered and AUC values were computed. ROC curves were plotted as shown in Figure 11 and Figure 12

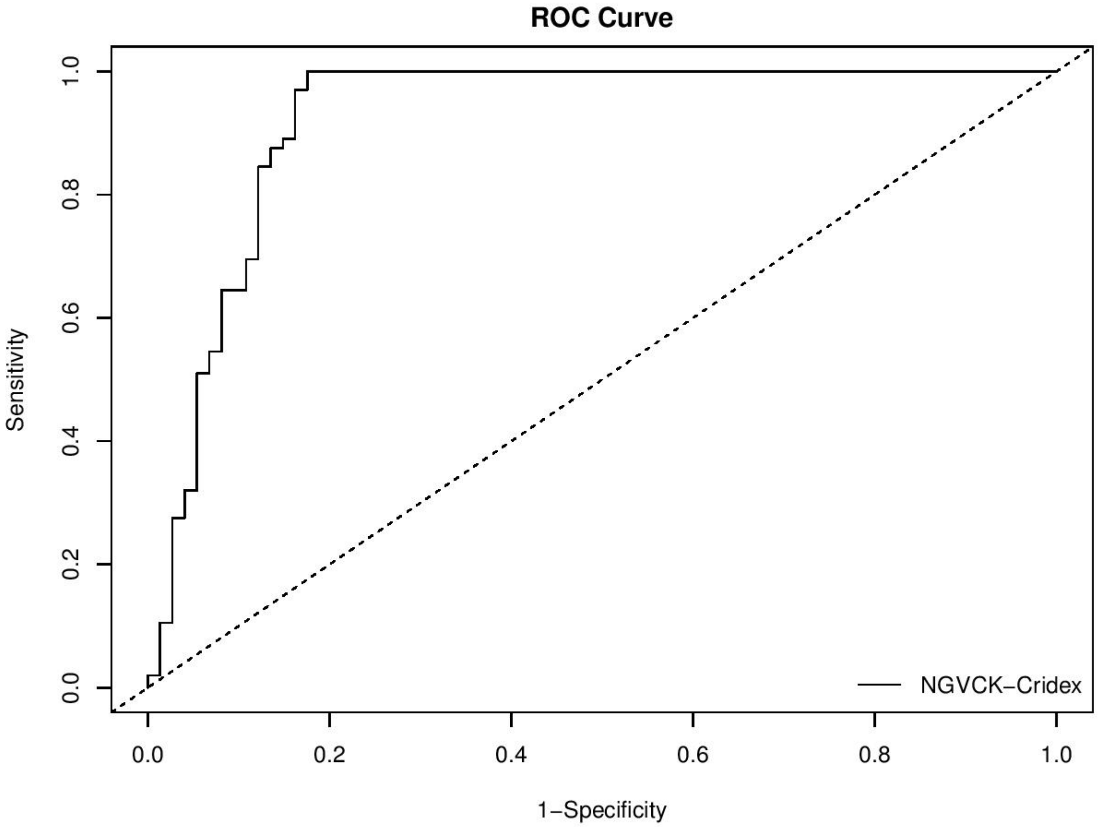

Figure 11: ROC curve for Cridex as test dataset

c	Training dataset	Testing dataset	AUC
26	NGVCK	Cridex	0.92351

Table 9: Parameters for Cridex as test dataset

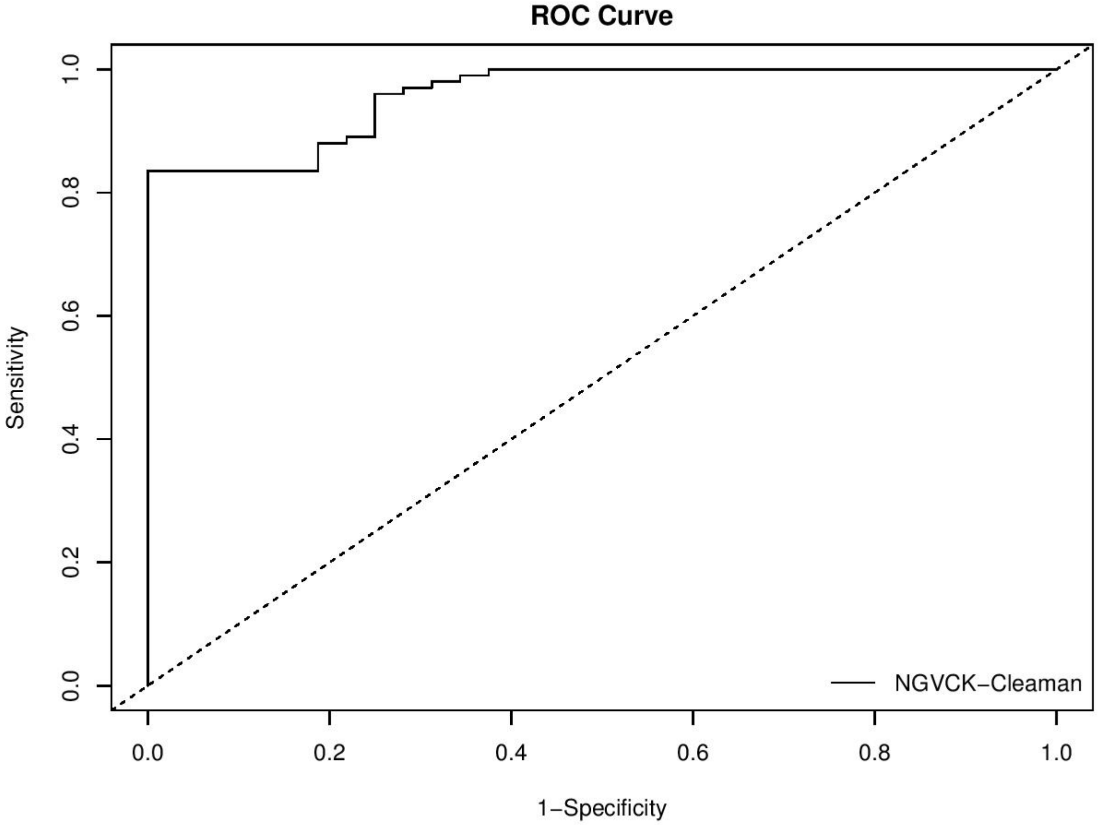

Figure 12: ROC curve for Cleaman as test dataset

c	Training dataset	Testing dataset	AUC
26	NGVCK	Cleaman	0.95875

Table 10: Parameters for Cleaman as test dataset

The AUC of all the scenarios considered were combined into the Table 11.

c	Training dataset	Testing dataset	AUC
26	NGVCK	Benign	0.92725
26	NGVCK	Cridex	0.92351
26	NGVCK	Cleaman	0.95875

Table 11: Results with different testing datasets

A bar graph has been plotted for the above generated AUC values as shown in

Figure 13.

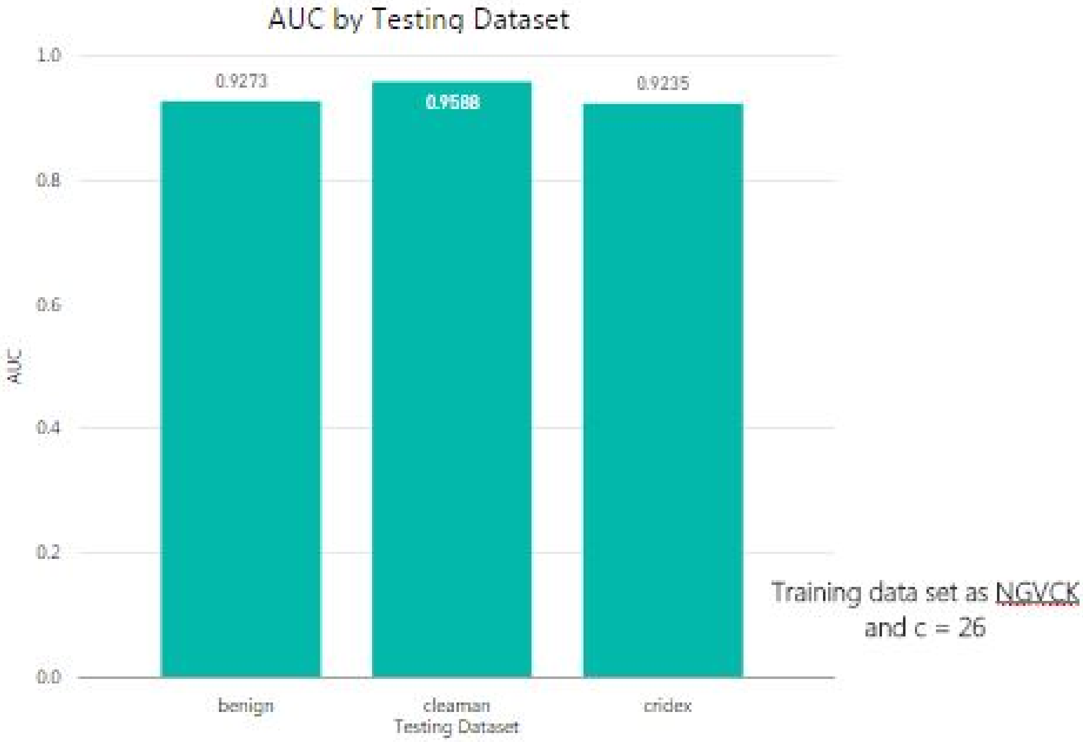

Figure 13: AUC by different testing datasets

As the IC was able to detect well with NGVCK as training dataset and benign as a testing dataset in the previous section, experiments were made to change the testing dataset and generate the IC. On observing the graph above, it can be said that all the three test datasets were detected really well when the training dataset was NGVCK.

4.4 Summary of results

On observing the above results and graphs, it can be said that IC scores are highly dependent on the choices made. If the values are chosen in an accurate way, IC can give better results, which helps in differentiating between two different malware families or between a malware family and a benign family. Please refer to appendix A to find the ROC curves for the results generated in the previous sections.

4.5 Comparison with other scores

To check the performance of IC, comparison of the results with different scores like the Vigenère cipher score and the simple substitution distance technique (SSD) were made. Table 12 contains values of the AUC generated for the different scores and a graph was plotted for these values as shown in Figure 14.

Malware Family	AUC using IC score	AUC using Vigenère Cryptanalysis score	AUC using SSD
NGVCK	0.92725	0.99	1
Zbot	0.4560744	0.9729	0.8664
Winwebsec	0.4600287	0.9996	0.8374
Smart HDD	0.6551471	0.9958	0.8855
Cridex	0.4351351	0.9458	0.5830
Security Shield	0.512931	0.5979	0.6290
Harebot	0.3188679	0.9057	0.5606
Zeroaccess	0.7189464	0.6588	N/A

Table 12: Comparison across different scores

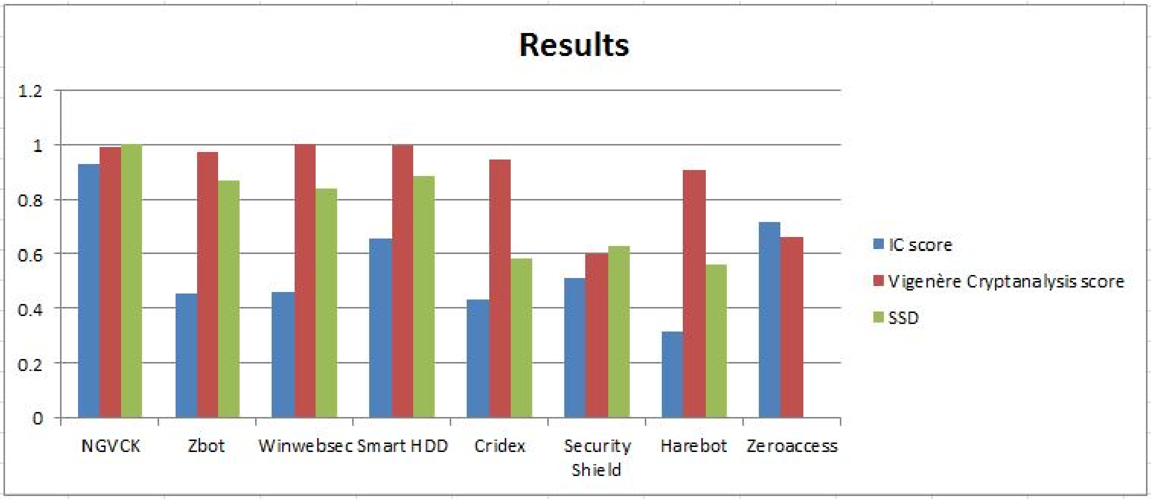

Figure 14: Comparing results with different scores

Based on the above graph, it can be observed that the IC does not perform as well as the other scores, as only one factor is involved in the calculation of the IC i.e. the frequency of opcodes.

CHAPTER 5

Observations

A few observations were made based on the results obtained in the previous chapter.

5.1 Observation 1:

IC is based on the frequency of opcodes and better results can be obtained only when the top c opcodes from both the training and the test dataset match.

To support the above observation, the percentage of matching opcodes for each malware family was computed across the benign set and plotted against the percentage of AUC.

Family	% of matching opcodes	% of AUC
NGVCK	56.96202532	92.725
Harebot	43.47826087	31.88679
Zeroaccess	24.75	71.89464
Zbot	30	45.60744
Security Shield	29.37853107	51.2931
Winwebsec	29.64071856	46.00287
Smart HDD	48.23529412	65.51471
Cridex	36.61417323	43.51351
Cleaman	31.63265306	41.32

Table 13: Percentage of matching opcodes

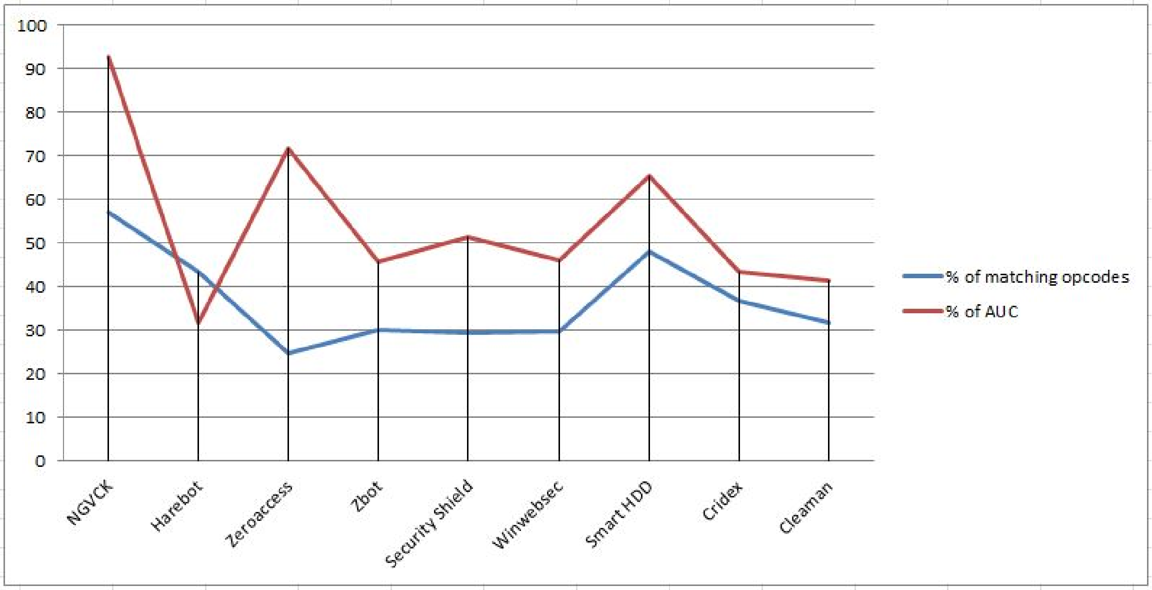

Figure 15: Percentage of matching opcodes

Figure 15 tells us that for most of the families, the aforementioned observation holds true. But, for families like Harebot and Zeroaccess this scenario is not true.

5.2 Observation 2:

IC can be based on the percentage of frequency of 'other' opcodes. To check whether this observation holds true, statistics were computed as given in Table 14.

Family	% frequency of 'other' opcodes	% of AUC
NGVCK	4.625928727	92.725
Harebot	6.265612543	31.88679
Zeroaccess	5.321108647	71.89464
Zbot	7.375108016	45.60744
Security Shield	4.183405127	51.2931
Winwebsec	3.787141968	46.00287
Smart HDD	0.289318786	65.51471
Cridex	6.522573331	43.51351
Cleaman	11.36468728	41.32

Table 14: Frequency percentage of 'other' opcodes

The above values were computed with 26 as the value of the normalizing coefficient. These values were plotted on a line graph, as shown in Figure 16, to check how the different families perform when the percentage of other opcodes across percentage of AUC was considered.

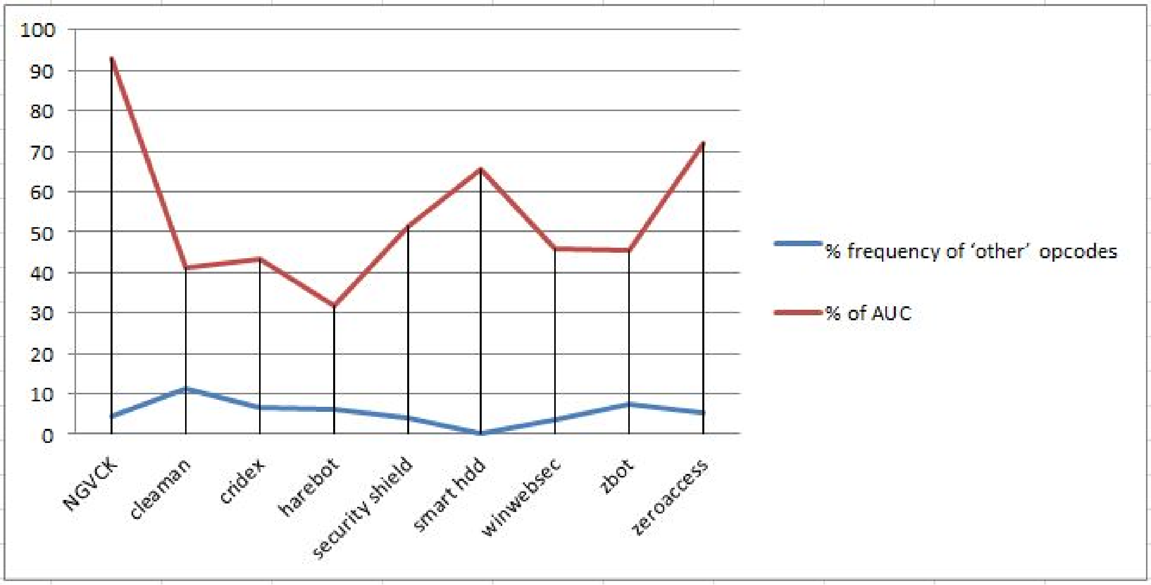

Figure 16: Frequency percentage of 'other' opcodes

Figure 16 does not exactly show the relation between the frequency of other opcodes to the values of AUC. This indicates that the observation does not hold true.

5.3 Observation 3:

IC can be based on the distribution of opcodes across the malware family. Distribution of a particular opcode was calculated by first calculating the frequency of that opcode in both the malware and the benign family. Then, the difference in their squares are calculated and then by square root of the final value is obtained. For example, if the frequency of an opcode, say "mov", in the malware family is x and its frequency in the benign family is y, then the distribution can be calculated as

$$\text{Opcode Distribution} = (x^2 - y^2)^{1/2} \tag{4}$$

Using equation 4, results were obtained and a line graph was plotted as shown in Figure 17. Here, the value of the normalizing coefficient is 26 and the test dataset is benign.

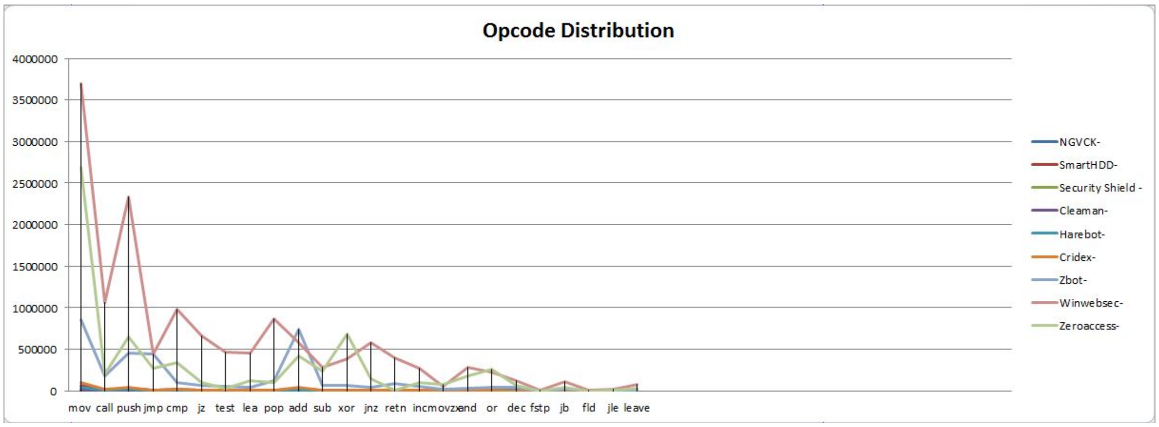

Figure 17: Opcode Distribution across families

By observing Figure 17, it can be said that the opcodes in NGVCK are distributed evenly. However, the opcodes in other families either start really high or really low, which denotes an uneven distribution. So, it can be said that the distribution of opcodes is an important factor for IC to perform better.

CHAPTER 6

Conclusion and Future Work

It can be said that the effectiveness of the IC score highly depends on the value of the normalizing coefficient, the training dataset and its corresponding test datasets. If these values are set properly, then the family of an unknown file can be effectively identified.

Based on the observations in the previous chapter, we can conclude that the value of IC is highly related to the distribution of opcodes across the different families.

Future work can focus on the calculation of "kappa" IC [5]. It is calculated using the null hypothesis, which states that at any given point in time, the expected no-correlation value is 1.0. Thus, the IC can be calculated as the coincidences observed divided by the coincidences expected. So, the term becomes

$$IC = \frac{\sum_{j=1}^{N}[a_j = b_j]}{\frac{N}{c}} \tag{5}$$

where, N is the common length of the two texts (say A and B), a_j is the j^{th} term of the text A, b_j is the j^{th} term of the text B and the bracketed term is 1 if true and 0 if false.

LIST OF REFERENCES

[1] C. Annachatre, T. H. Austin, M. Stamp, Hidden Markov Models for Malware Classification, *Journal of Computer Virology and Hacking Techniques*, 11(2):59–73, May 2015

[2] J. Aycock, *Computer Viruses and Malware, Advances in Information Security*, Springer-Verlag, New York, 2006

[3] A. P. Bradley, The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms, *Journal Pattern Recognition*, 30(7):1145–1159, 1997

[4] S. Deshmukh, Vigenère Score for Malware Detection, Master's report, Department of Computer Science, San Jose State University, 2016, `http://scholarworks.sjsu.edu/etd_projects/487/`

[5] W. F. Friedman, The index of coincidence and its applications in cryptology, *Department of Ciphers. Publ 22. Geneva, Illinois, USA: Riverbank Laboratories*, 2014

[6] H. Rana, M. Stamp, Hunting for Pirated Software Using Metamorphic Analysis, *Information Security Journal: A Global Prospective*, 2014

[7] G. Shanmugam, Vigenère Simple Substitution Distance and Metamorphic Detection, Department of Computer Science, San Jose State University, 2012, `http://scholarworks.sjsu.edu/etd_projects/270/`

[8] G. Shanmugam, R. M. Low, and M. Stamp, Simple substitution distance and metamorphic detection, *Journal of Computer Virology and Hacking Techniques*, 9(3):159–170, 2013

[9] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley, 2006

[10] M. Stamp, *Information Security: Principles and Practice*, second edition, Wiley, 2011

[11] S. Sing, *The code book: the science of secrecy from ancient Egypt to quantum cryptography*, Anchor, 2011

[12] S. Srinivasan, SSCT Score for Malware Detection, Master's report, Department of Computer Science, San Jose State University, 2015, `http://scholarworks.sjsu.edu/etd_projects/444/`

[13] Symantec Annual Security Report 2008, http://www.realwire.com/releases/symantec-announces-messagelabs-intelligence-2008-annual-security-report

[14] Understanding and Managing Polymorphic viruses https://www.symantec.com/avcenter/reference/striker.pdf

[15] S. Vemparala, Malware Detection Using Dynamic Analysis,Master's report, Department of Computer Science, San Jose State University, 2015, http://scholarworks.sjsu.edu/etd_projects/403/

APPENDIX

Generated ROC Curves

c	26
train	NGVCK
test	Benign

Table A.15: Parameters NGVCK-Benign-26

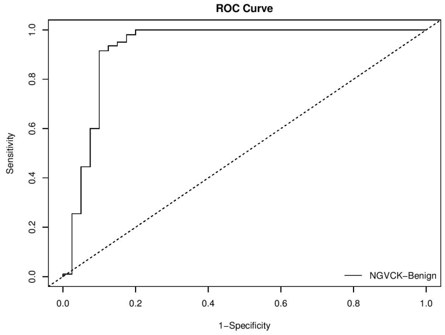

Figure A.18: ROC Curve NGVCK-Benign-26

c	26
train	NGVCK
test	Cleaman

Table A.16: Parameters NGVCK-Cleaman-26

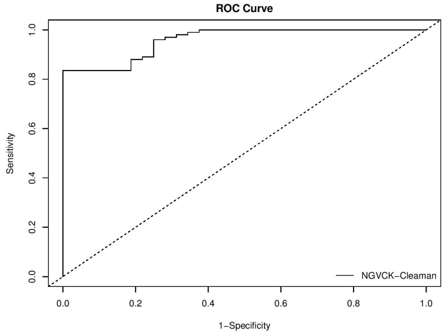

Figure A.19: ROC Curve NGVCK-Cleaman-26

c	26
train	NGVCK
test	Cridex

Table A.17: Parameters NGVCK-Cridex-26

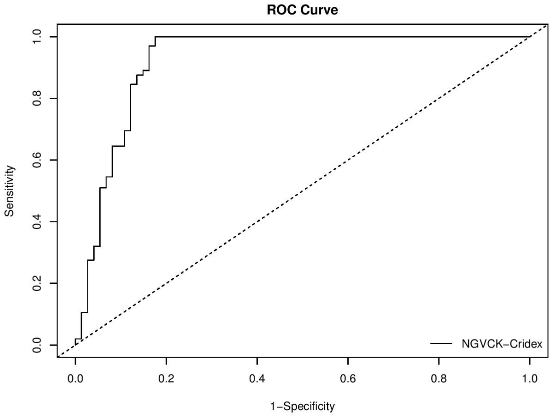

Figure A.20: ROC Curve NGVCK-Cridex-26

c	26
train	ZeroAccess
test	Benign

Table A.18: Parameters ZeroAccess-Benign-26

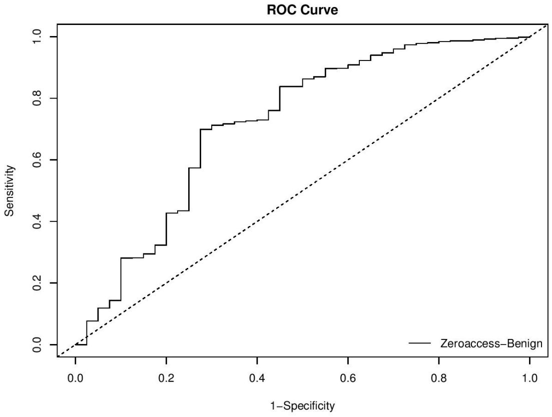

Figure A.21: ROC Curve ZeroAccess-Benign-26

c	26
train	Smart HDD
test	Benign

Table A.19: Parameters Smart HDD-Benign-26

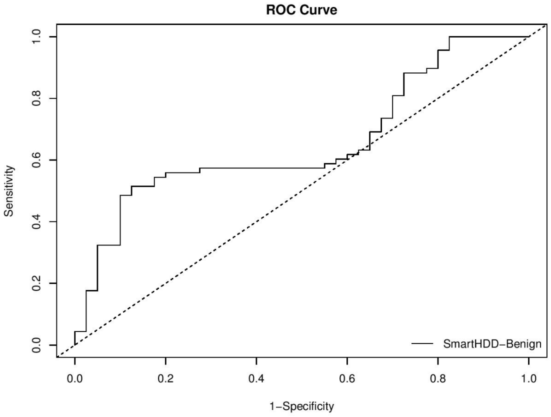

Figure A.22: ROC Curve Smart HDD-Benign-26

c	26
train	Security Shield
test	Benign

Table A.20: Parameters Security Shield-Benign-26

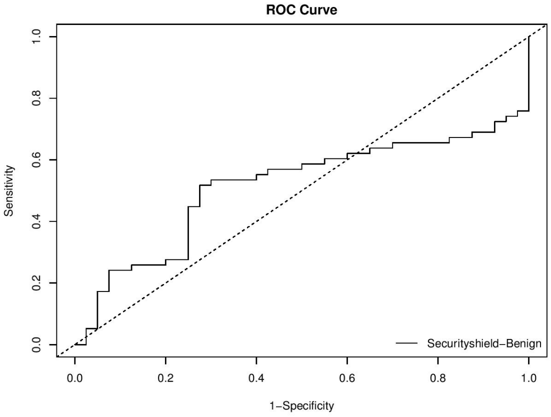

Figure A.23: ROC Curve Security Shield-Benign-26

c	26
train	Winwebsec
test	Benign

Table A.21: Parameters Winwebsec-Benign-26

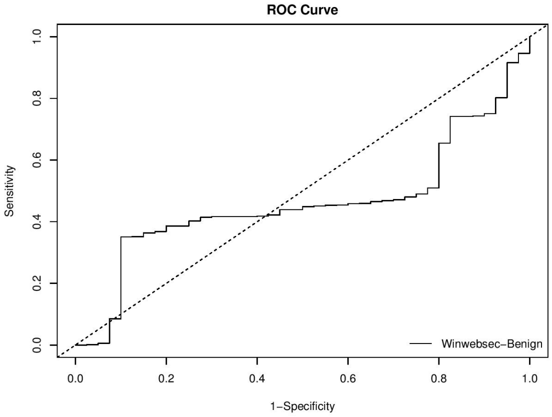

Figure A.24: ROC Curve Winwebsec-Benign-26

c	26
train	Zbot
test	Benign

Table A.22: Parameters Zbot-Benign-26

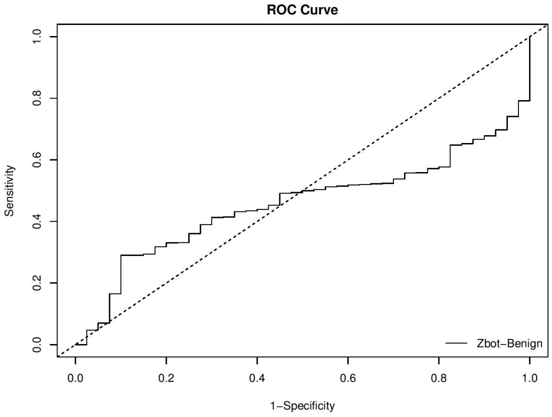

Figure A.25: ROC Curve Zbot-Benign-26

c	26
train	Cridex
test	Benign

Table A.23: Parameters Cridex-Benign-26

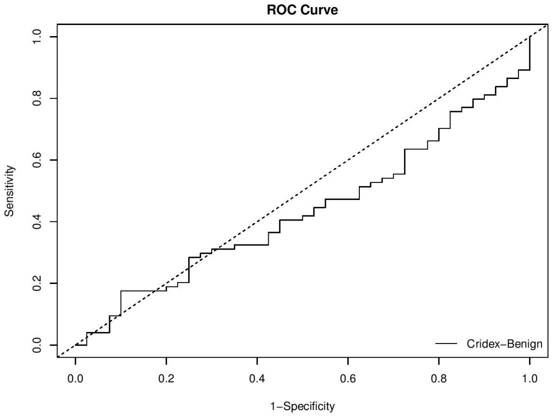

Figure A.26: ROC Curve Cridex-Benign-26

c	26
train	Cleaman
test	Benign

Table A.24: Parameters Cleaman-Benign-26

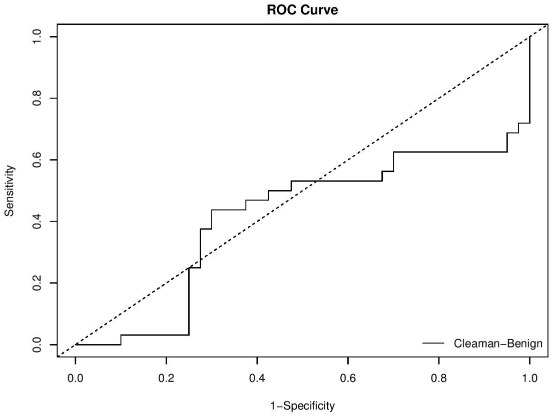

Figure A.27: ROC Curve Cleaman-Benign-26

c	26
train	Harebot
test	Benign

Table A.25: Parameters Harebot-Benign-26

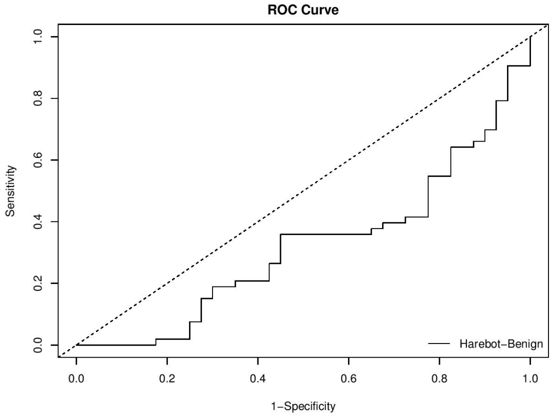

Figure A.28: ROC Curve Harebot-Benign-26

c	79
train	NGVCK
test	Benign

Table A.26: Parameters NGVCK-Benign-79

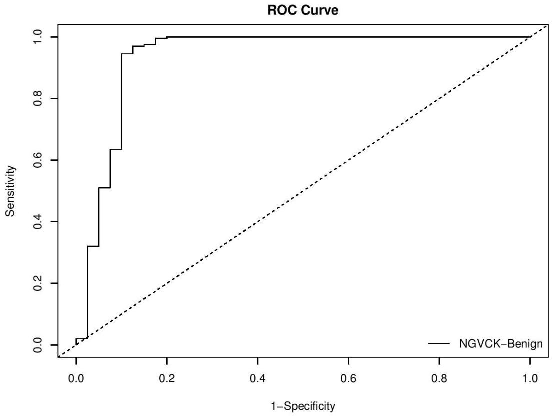

Figure A.29: ROC Curve NGVCK-Benign-79

c	79
train	NGVCK
test	Zbot

Table A.27: Parameters NGVCK-Zbot-79

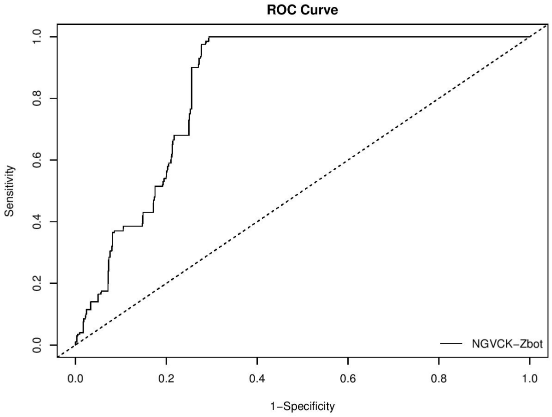

Figure A.30: ROC Curve NGVCK-Zbot-79

c	63
train	NGVCK
test	Benign

Table A.28: Parameters NGVCK-Benign-63

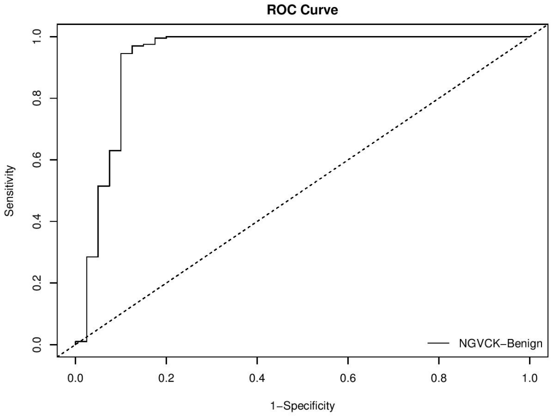

Figure A.31: ROC Curve NGVCK-Benign-63

c	63
train	NGVCK
test	Security Shield

Table A.29: Parameters NGVCK-Security Shield-63

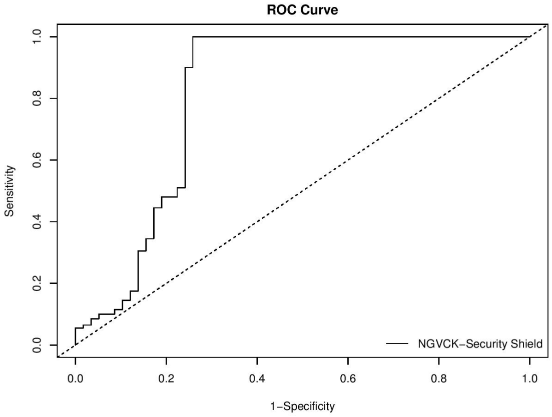

Figure A.32: ROC Curve NGVCK-Security Shield-63

c	50
train	NGVCK
test	Benign

Table A.30: Parameters NGVCK-Benign-50

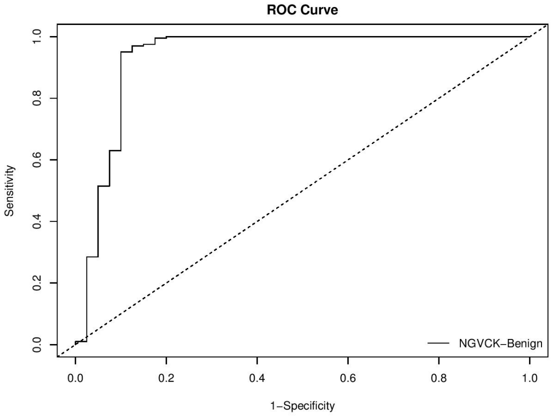

Figure A.33: ROC Curve NGVCK-Benign-50

c	50
train	NGVCK
test	Winwebsec

Table A.31: Parameters NGVCK-Winwebsec-50

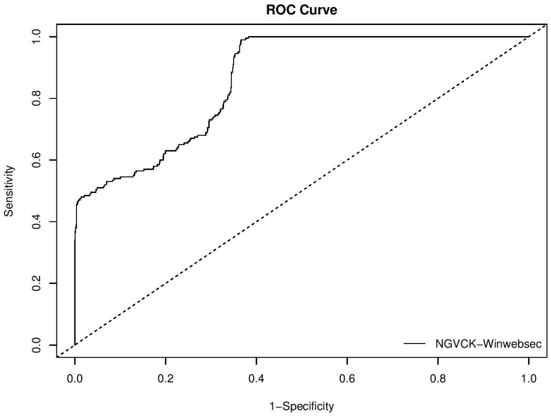

Figure A.34: ROC Curve NGVCK-Winwebsec-50

c	45
train	NGVCK
test	Benign

Table A.32: Parameters NGVCK-Benign-45

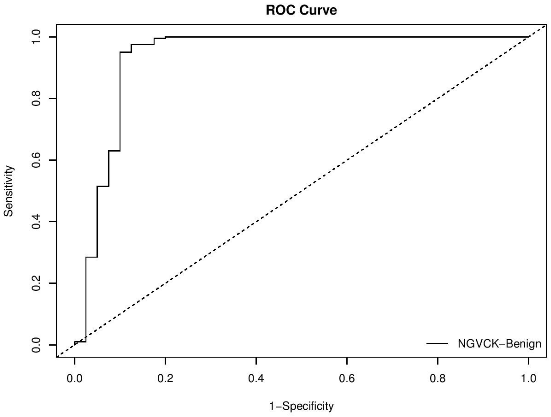

Figure A.35: ROC Curve NGVCK-Benign-45

c	45
train	NGVCK
test	Smart HDD

Table A.33: Parameters NGVCK-Smart HDD-45

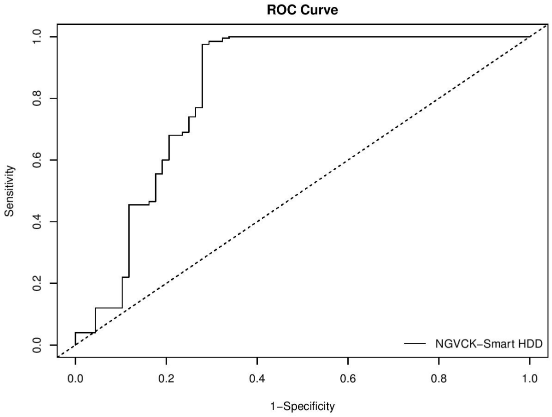

Figure A.36: ROC Curve NGVCK-SmartHDD-45

c	34
train	NGVCK
test	Benign

Table A.34: Parameters NGVCK-Benign-34

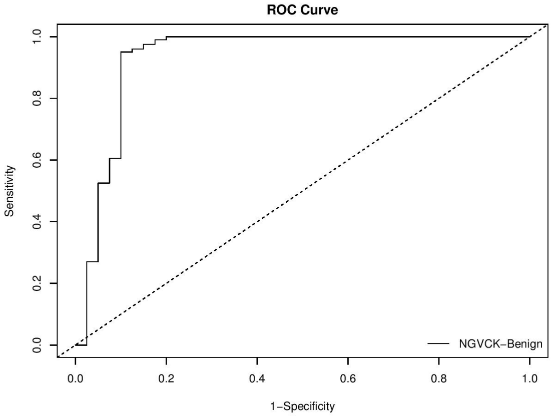

Figure A.37: ROC Curve NGVCK-Benign-34

c	34
train	NGVCK
test	ZeroAccess

Table A.35: Parameters NGVCK-Zeroaccess-34

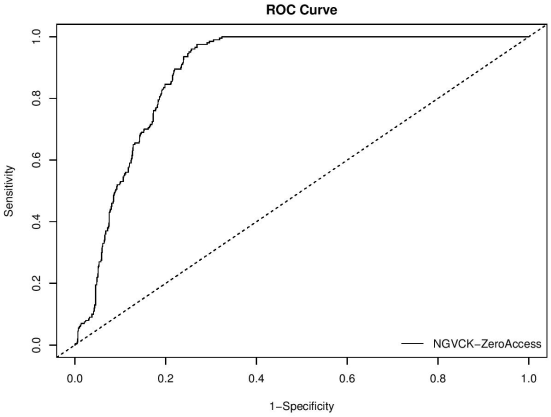

Figure A.38: ROC Curve NGVCK-Zeroaccess-26

c	12
train	NGVCK
test	Benign

Table A.36: Parameters NGVCK-Benign-12

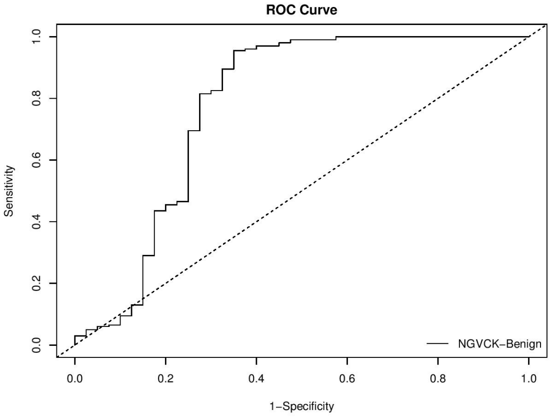

Figure A.39: ROC Curve NGVCK-Benign-12

c	12
train	NGVCK
test	Harebot

Table A.37: Parameters NGVCK-Harebot-12

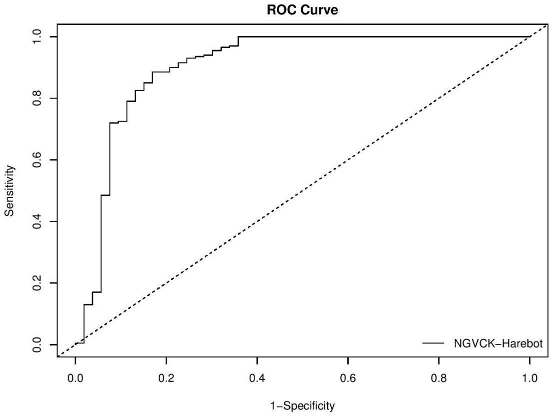

Figure A.40: ROC Curve NGVCK-Harebot-12

c	6
train	NGVCK
test	Benign

Table A.38: Parameters NGVCK-Benign-6

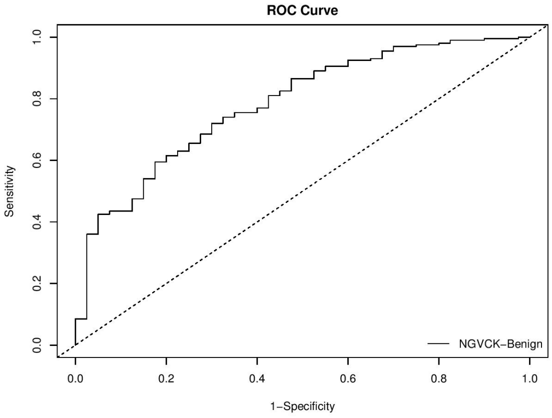

Figure A.41: ROC Curve NGVCK-Benign-6