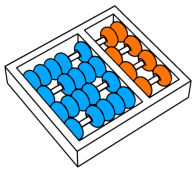
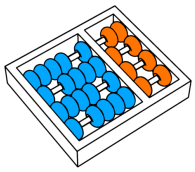


Alexandre Or Cansian Baruque

"Classifying Malware Using Dynamic Analysis and Profile Hidden Markov Models"

CAMPINAS
2015



University of Campinas
Institute of Computing

Alexandre Or Cansian Baruque

"Classifying Malware Using Dynamic Analysis and Profile Hidden Markov Models"

Supervisor(s)
Prof. Dr. Paulo Lício de Geus[1]
Dr. André Ricardo Abed Grégio[2]

Term Paper present to the Commission of Graduation of the Institute of Computing.

This volume correspond to the final version of the term paper defended by Alexandre Or Cansian Baruque, under the supervision of Prof. Dr. Paulo Lício de Geus.

[1]Supervisor's signature

[2]Supervisor's signature

CAMPINAS
2015

Abstract

Malicious software, also know as malware, is a persistent threat to the security of information systems. The continuous development of new exploits, attack methods and malware samples creates a need for security analysts to stay up to date with current threats.

To aid in this task, tools such as dynamic analysis systems are used to extract information from malware samples. However, the volume of information extracted by such tools may be too large for manual analysis or inspection. In this work, we use Profile Hidden Markov Models to create statistical models using supervised learning. The models are generated by applying Multiple Sequence Alignment algorithm to the sequence of actions executed by the malware sample inside a dynamic analysis environment.

A total of 606 malware samples were used, which we split into 301 for training the models and 305 for testing these created models. Those classification models allow the labeling of unknown samples, as well as uncovering information regarding malware operation. When tested, the models performed well, since they obtained high F-Scores. We were also able to identify similar behavior among different malware families by comparing the labeling results present in the confusion matrix. The results obtained indicate that further research in this area can provide promising insights for defense against malware attacks.

Keywords: Malware Analysis, Malware Classification, Malware Behavior, Malicious Software, Dynamic Analysis, Hidden Markov Models, Profile Hidden Markov Models, Multiple Sequence Alignment, Supervised Learning, Machine Learning

Acknowledgments

I would like to thank Prof. Paulo Lícios de Geus, without whom it would not be possible complete this work.

I would like express my gratitude to André Ricardo Abed Grégio, for helping me along this path with all the knowledge and experience shared.

I would also like thank my colleagues at LASCA laboratory that provided me with fruitful discussions and insights.

Finally, I thank my friends and family for always supporting and encouraging me.

"Security is just like driving on the highway – anyone going slower than you is an idiot, while anyone driving faster than you is a crazy person"

Konstantin Ryabitsev

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Malicious software, or malware, are programs developed with the intention of executing malicious, unknown and damaging activities. They are a persistent threat to the security of information systems and its users.

To be able to develop better counter-measures for this threat, first we must understand what we are trying to defend against. Malware analysis is the main form of obtaining information regarding malware samples. Another important step is the classification of malware samples into families, so as to group them together based on some features and to discover unknown, undetected software that are similar to known malware.

To help in this task, we can make use of tools such as dynamic analysis systems, which allow us to extract information about malware execution. However, the volume of information generated by such tools can be too large for manual analysis or inspection.

Therefore, as the analysis systems increase their complexity, so does the amount and complexity of the data collected by these systems. In this work, we intend to leverage the modeling capabilities of Profile Hidden Markov Models to create classificatory models for different malware families. To create these models, we extract behavioral information from the observed actions of real world malware samples, monitored while executed inside a dynamic analysis environment. The sequence of actions produced after the analysis is then used to train our statistical models.

Training the models consists in using antivirus labels to group malware samples, each sample containing the sequence of action taken by the malicious program. The sequences are then aligned using Multiple Sequence Alignment algorithm, this alignment allows us to extract the information required to create the Profile Hidden Markov Models.

This document is organized as follows: in Chapter 2, we introduce the concepts and background knowledge used in this work, and the related works; in Chapter 3, we explain in details how the models were created, as well as the method used for assigning labels to the test samples; in Chapter 4, we discuss and evaluate the results obtained by the

produced models, and finally, in Chapter 5, we conclude this text by pointing out some of the limitations encountered and suggesting future works.

Chapter 2

Background

2.1 Malware Analysis

According to Sikorski [15], malware analysis is the art of dissecting the malware to understand its functioning. In the following subsections we briefly introduce the two main approaches to malware analysis, both with its advantages and disadvantages.

2.1.1 Static Analysis

Commonly used as a first step in analyzing malicious software, static analysis consists in examining the malicious file without executing it. This type of analysis can confirm that the file is indeed malicious and also obtain information about its functionality, libraries required for its execution, or defensive mechanisms used by the malware to evade detection and analysis.

This type of analysis is commonly used for the creation of signatures for antivirus and intrusion detection systems, in which this kind of approach is preferred due to its low overhead cost, since it does not need to execute the binary in a controlled environment.

Due to its simplicity, it is also easy for the malware creators to defend against static analysis techniques, either by the insertion of dead-code, application of instruction/function reordering, and/or code transposition to obfuscate the binary code, or the use of packers to encrypt/compress the binary hiding its information [25].

Static Analysis of a given binary can be classified as NP-Hard problem. Moser Et al. [16] showed that by using opaque constants to generate transformations that obfuscate a binary program would cause Static Analysis to be NP-Hard Problem.

2.1.2 Dynamic Analysis

Dynamic analysis consists of executing a program inside a controlled environment that can be either a virtual machine, an emulator, or a "bare-metal" system. During its execution, we can extract information through the monitoring of the system calls and other operating system interactions, such as API function calls, memory-mappings or Registry values accessed and modified by the malicious program.

One important factor is allowing the sample to have (limited) access to the Internet, so it can download additional components, receive information from a malicious C&C server, and so on.

Dynamic analysis allows us to capture the malware behavior directly which avoids the packers issues, but as all arms-race we see also that malware builders develop new tools to combat this analysis method [9]. Common evasion tactics are checking if debug hooks are used in the program, search for registers that are specific for virtual machine or simply just waiting dormant on the system for a long time to make impractical for the analyst to make large scale analysis. Bare-metal machines are harder to detect but the malware can also check if there is user interaction or files.

In this thesis we use Dynamic Analysis to extract information from multiple malware samples, this information is then passed to statistical models to create classifiers. More specifically, we extract the malware behavior in a log of its sequence of high level actions in the system. This information was extracted using BehEMOT [12].

2.2 Markov Process

A Markov Process or Markov Chain is a stochastic model for a random system that change its states according to fixed probabilities.

Memorylessness is an important property of a Markov Process. It implies that the probability distribution of future states depends only on the current state and has no dependency with the sequence of states prior the current state [17].

This process which depends only on the current state of the system is defined as order-one Markov Process. Higher order Markov processes are those in which the current event depends on more than one previous event. This does not limit the modeling capabilities due to its equivalence with order one [2].

2.3 Hidden Markov Models

The Hidden Markov Model is a Markov Process where we are unable to directly observe the state of the system, as we can see in Figure 2.1. We can only observe the events and

"secondary" effects caused by the state. Each state has a fixed probability of "emitting" an event to be observed.

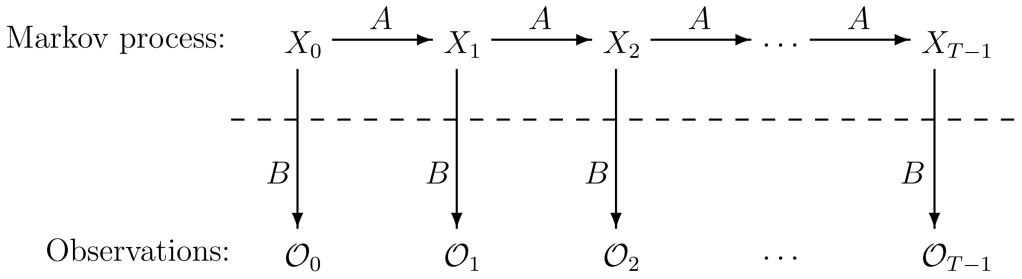


Figure 2.1: Markov Process and its observations

In the following subsections we introduce to the reader the basic notation used and also explore some classical problems involved with Hidden Markov Models as well as the useful applications following that.

2.3.1 Notation

$T =$ length of the observation sequence

$N =$ number of states in the model

$M =$ number of observation symbols

$Q = \{q_0, q_1, ..., q_{n-1}\} =$ distinct states of the markov process

$V = \{0, 1, ..., M-1\} =$ set of possible observations

$A =$ state transition probabilities NxN matrix

$B =$ observation probability MxN matrix

$\pi =$ initial state distribution

$O = (O_1, O_2, ..., O_{T-1}) =$ observation sequence.

$\lambda = (A, B, \pi) =$ A Hidden Markov Model defined by the tuple (A, B, π).

2.3.2 Classical Problems for HMM

In the literature for Hidden Markov Models, we have the following three main problems as show by Rabiner [19].

1. Given the model $\lambda = (A, B, \pi)$ and a sequence of observations O, find $P(O|\lambda)$. Here, we want to determine the likelihood of the observed sequence O, given the model.

2. Given $\lambda = (A, B, \pi)$ and an observation sequence O, find an optimal state sequence for the underlying Markov process. In other words, we want to uncover the hidden part of the Hidden Markov Model.

3. Given an observation sequence O and the dimensions N and M, find the model $\lambda = (A, B, \pi)$ that maximizes the probability of O. Alternatively, we can view this as a (discrete) hill climb on the parameter space represented by A, B and π.

Problems 1 and 3 are the focus of our interest in this monograph.

Problem 3 allow us to create Hidden Markov Models based on a sequence of observations, this is equivalent to training a model to best fit the observed data.

Problem 1 allow us to obtain a score based on how well a certain observation sequence O matches with a model λ, with this score we are able to determine how well a model represents the given observation sequence.

Both problems have efficient methods of solving them by making use of the properties of Markov Processes and Hidden Markov Models, the Viterbi algorithm[11] is one of these methods. It is also important to define what is "most likely" due to its impact in the definitions of the algorithm, see [23] for more details.

2.4 Profile Hidden Markov Models

One of the issues encountered in this work with HMM is it assumption that we observe a single long sequence of events, but in our case we have multiple observation sequences each from a malware sample.

A simple solution to this issue would be to concatenate the string of events of all samples into one long sequence, however this introduces a periodic artificial information in our models in each of the junction of the strings implying that the final state leads to the initial state thus creating a cyclical nature in the model, which could impact the results.

There are better solutions for multiple sequence of observations. One of them is the Profile Hidden Markov Model (PHMM). A PHMM is a specific formulation of a standard HMM that makes explicit use of positional information contained in the observation sequences [2]. PHMM is a strongly linear left-right model while HMM is not [7].

PHMM are used in bioinformatics in DNA sequencing, where the comparison of the sequences may contain modifications such as insertions and deletions that occur naturally with mutation during the process of evolution. This behavior is similar to metamorphic malware that go through a evolutionary process [6].

In Figure 2.2 we can see the structure of the states in a PHMM. There are three type of states each trying to model an specific part of the sequence alignment. The Match state represents each symbol in a column of a MSA. The Insert state models after random insertions of symbols that can occur due to mutations in the sequences. The Deletion state models sections of a sequence that can be removed.

To obtain the probabilities in the state transition matrix of the PHMM we can extract this information from the frequency of the symbols in each column of a Multiple Sequence Alignment [2]. Thus creating a MSA is equivalent to training a PHMM.

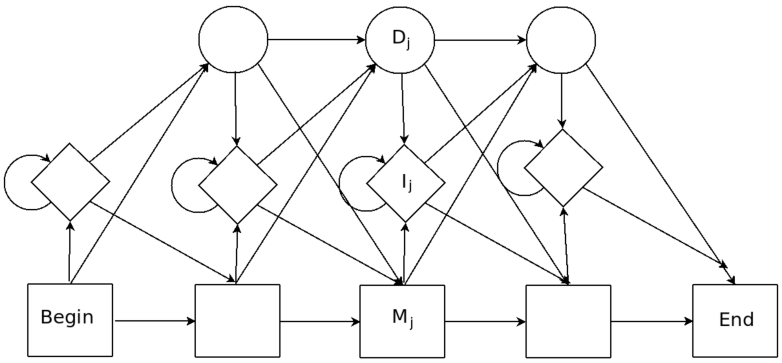


Figure 2.2: Profile Hidden Markov Model states, Squares are match, Diamonds are insert, and Circles are deletion.

2.5 Multiple Sequence Alignment

An important concept for Profile Hidden Markov Models is the use of positional information to construct its models this information originates from multiple sequence alignment.

In DNA sequencing, we can align multiple sequences of genes that are known to have some significant biological relationship. The resulting alignment can then be used to determine whether an unknown sequence might be related to the sequences that comprise the sequence alignment.

In bioinformatics applications, the purpose of aligning sequences is to look for evidence that the sequences diverged from a common ancestor by a process of mutation and natural selection [6].

To construct a multiple sequence alignment we must first determine a pairwise sequence alignment of two sequences, which yields a pair of sequences of equal length that captures the difference between the two original sequences by inserting gaps represented by '-' in Figure 2.1 we can see an example of this alignment.

The global alignment is an alignment such that the matches are maximized and the insertions/deletions are minimized [18].

Table 2.1: An example of a pairwise sequence alignment.

```
Original Sequences:
EMEKECNEMEDEDEDMDMEMEMEMEMENPNMNMEGMNMFMMEMMEMNPNMNMEMEEGM
EMEKECMEDEDEDMEMEMEMEMEMEMENPNMNMEGMNMEMMENMEMENPNMNMEECM
Aligned Sequences:
EMEKECNEMEDEDEDMDMEMEMEMEMEN----PNMNMEGMNMF-MMEMMEM-NPNMNMEMEEG-M
EMEKEC--MEDEDED--MEMEMEMEMEMEMENPNMNMEGMNM-EMMENMEMENPNMNME-E--CM
```

Table 2.2: An example of a multiple sequence alignment

```
EMEKECNEMEDEDEDMDMEMEMEMEMPNMNDMEMEMENPN--------------MNMEMFEEGMNMLM------------
EMEKECMEDEDEDMEMEMEMEMENMNDEMEMEMNPNM-----------------NMEMEMEMNGDMLM------------
EMEKECMEDEDEDMEMEMEMEMENMNDMEMEMEMENPNMNMEGMNMFMEGMEMGMEMSEMESTSVGCMNGDFRMVEHMLM
EMEKECMEDEDEDMEMEMEMEMENMNDMEMEMEMENPNMNMEGMNMEMFE----MEGMGMEMNGDMLM------------
EMEKECMEDEDEDMEMEMEMEMENMNDMEMEMEMNP------------------NMNMEMEEGMCMNGDNMLM-------
EMEKECMEDEDEDMEMEMEMEMENMNDMEMEMEME-------------------NPNMNMEECMNGDMNLM---------
```

2.5.1 Encoding

To make it possible for the behavior logs to be passed on to the Multiple Sequence Alignment software, it is required to encode the information in a supported format. Is this case, letters which in the normal usage are intended for representing RNA proteins. Here we assign each action with a corresponding letter. It is important to note that the specific information of the malware action is ignored. This is because we are aiming to profile the general behavior of the malware and not its specific effects on the system. In the table 2.3 we can see the encoding used in this thesis.

Another step taken during the encoding of behavior logs is the remotion of repeated action in sequence. The idea behind is that, when creating a model for the behavior of the Malware, the fact that the sample executed e.g. a WRITE FILE action is encoded, rather than how many times in a row it executed that action. This permits the generated models to be more generic and allows us to create better MSA.

Table 2.3: Enconding used for the behaviour actions

Action	**Code**
QUERY MUTEX	A
CREATE MUTEX	C
RELEASE MUTEX	D
READ FILE	E
WRITE FILE	F
CREATE FILE	G
DELETE FILE	H
CREATE PROCESS	I
OPEN PROCESS	K
TERMINATE PROCESS	L
READ REGISTRY	M
WRITE REGISTRY	N
CREATE REGISTRY	P
DELETE REGISTRY	Q
RECEIVE NET	R
SEND NET	S
CONNECT NET	T
DISCONNECT NET	V
WRITE MEMORY	W
LOAD DRIVER	Y

2.6 Kernel Density Estimation

When trying to align sequences of different lengths, it is inevitable that the resulting alignment has gaps inserted, however the problem is that the more gaps we insert in our alignment, the more generic the resulting PHMM will be [2]. In hopes to limit the number of the gaps inserted a clustering of the sequences based on their length was done.

To do such clustering, specific 1-Dimensional clustering methods where used which make use of the restrictions we have for a better clustering. The method chosen for this task was the Kernel Density Estimation, which estimates a continuous density probability function out of discrete data, in our case a histogram with the lengths of all given samples that we want to align together.

$$\rho_K(y) = \sum_{i=1}^{N} K((y - x_i)/h) \tag{2.1}$$

Equation 2.1 is used to determine the probability function $\rho_K(y)$ at a given point y, $x_i(i = 1 \cdots N)$ are our data points in this case the length of our observation sequences.

h is the bandwidth parameter that acts as a smoothing parameter, controlling the trade off between bias and variance in the result. A large bandwidth leads to a very smooth (i.e. high-bias) density distribution. A small bandwidth leads to an unsmooth

(i.e. high-variance) density distribution [21].

The kernel, $K(x;h)$, is a positive function, which is controlled by the bandwidth parameter h. The Figure 2.4 shows us the different forms of kernels supported by the *scikit-learn* library, in this thesis we chose to use the gaussian kernel.

In figure 2.3 we can see and example of the estimations of the distribution probability function of the samples labeled Worm-Allapple.

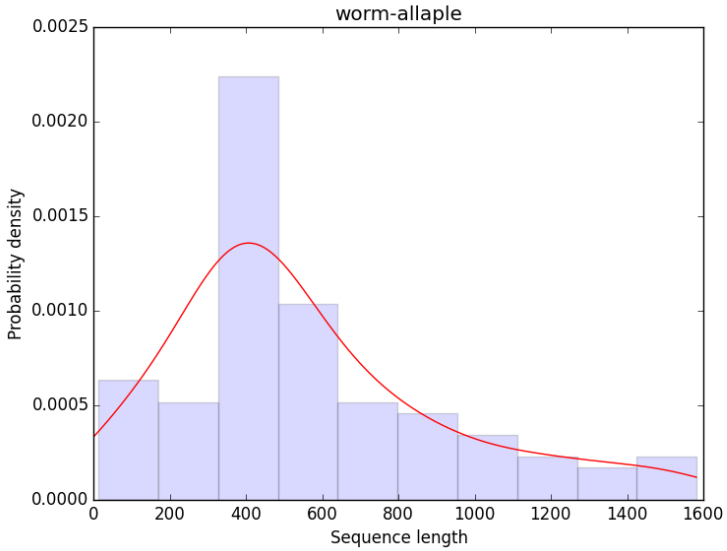


Figure 2.3: Kernel Density Estimation on the histogram of the length of Worm-Allaple samples.

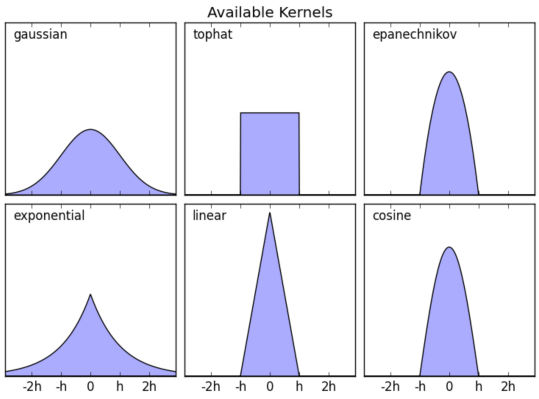


Figure 2.4: Kernels supported by the scikit-learn library.

2.7 Evaluation Metrics

In this section we briefly define the metrics used to evaluated the results obtained by our classificatory models. We chose to use the same metrics as Saradha [20].

2.7.1 Notation

$$\begin{aligned} tp &= \text{True Positive} \\ fp &= \text{False Positive} \\ tn &= \text{True Negative} \\ fn &= \text{False Negative} \end{aligned}$$

2.7.2 Accuracy

Accuracy is the ratio of the number of total correctly classified samples to the total number of samples in the dataset. This measure is not sufficient by it self, since it depends on the total number of samples the results of accuracy can be skewed by the proportion of positive samples over the negative samples.

$$\text{Accuracy} = \frac{tp + tn}{tp + fp + tn + fn} \tag{2.2}$$

2.7.3 Precision

Precision or Positive Predictive Value, is the ratio of correctly classified as positive samples by the total number of samples classified as positive. It represents how reliable is a positive result form the classificator.

$$\text{Precision} = \frac{tp}{tp + fp} \tag{2.3}$$

2.7.4 Recall

Recall or True Positive Rate, is the ratio of true positive classifications by the total number of positive samples in the dataset. It represents how well a classificator covers the dataset.

$$\text{Recall} = \frac{tp}{tp + fn} \tag{2.4}$$

2.7.5 F-Score

F-Score of F-Measure, is the harmonic mean of the precision and recall. It is useful since to have a good classificator we want both the recall and precision to be high, which is represented by this metric.

$$\text{F-Score} = \frac{2.(\text{Precision.Recall})}{(\text{Precision} + \text{Recall})} \tag{2.5}$$

2.8 Related Works

Annachhatre Et al. [1] considered the problems regarding the classification of malware based on Hidden Markov Models. Multiple HMM were trained, and then separated 8.000 samples into clusters based on their scores. The samples used for training and testing where different, showing that HMM approach can be effective at classifying unknown malware.

Saradha R [20] analyzed in detail the use of Profile Hidden Markov Models to malware clustering and classification. They found that the use of a small sample for training could prove helpful to early detection and classification of malware families. A phylogenetic analysis of malware families was also done, it explored the relationship between the models, and how code reuse/borrowing from malware creators create at network of derivation relationship.

Kalbhor Et al. [14] presented the concept of dueling Hidden Markov Models, where different HMM would be competing and the one with highest score would get to assign the label to the sample. They also showed a *tiered HMM strategy* to reduce the performance overhead cost of dueling HMM by passing an potential malware file through multiple layers of trained HMM in order to minimize the number of files that go through the dueling phase.

Attaluri Et al. [2] applied Profile Hidden Markov Models to polymorphic malware at instruction level and found encouraging results regarding the the utility of PHMM for detecting metamorphic virus variants generated from virus construction kits. They developed models based on the assembly code instructions finding that PHMM is well suited for certain types of metamorphic malware.

Austin Et al. [3] researched the meaning behind the hidden states of Hidden Markov Models created from assembly code generated by four different compilers, three virus construction kits, a metamorphic virus, and hand-written code. They note the similarities and differences in the hidden states from each source.

Chapter 3

Creating Profile Hidden Markov Models

In this Chapter we go in details on how the Profile Hidden Markov Models were created. Starting from data collection and extraction of behavior sequences from the binary files to generating Profile Hidden Markov Models, and finally, how to use these models to test unknown samples.

3.1 Data collection

The dataset used consists of 606 know malware executable samples collected from multiple sources: honeypots, spam/phishing emails, collaborative repositories with other researchers.

3.2 Antivirus Labels

Antivirus labels were used as the ground truth used to train and create the models, each sample in our data set was submitted to the Virustotal [24] to obtain labels for diverse antivirus programs. We chose the Avira [4] due to having a good score in the in the Virus Bulleting RAP tests [5] combined with labels that in some way explain the behavior o the Malware.

Labels that provide no information about the behavior of the malware were ignored, such as generic, heuristic. Labels that identify a packer where also discard since any behavior can be executed from them and thus we have no grantee that Malwares labeled with theses same labels have any actual similarity.

3.3 BehEMOT

The Dynamic Analysis Environment, BehEMOT [12] was used to extract the Malware Behavior. Each malware sample was executed inside the analysis environment resulting in a corresponding behavior log which represent the sequence of actions executed by the malware.

In Table 3.1 we see an example of the log files used to create malware and containing information of what was performed and at what resource of the analysis system.

Table 3.1: Snippet of behavior log extracted from a malware sample using BeHEMOT.

```
01d23d9.exe;READ;REGISTRY;reg_string:\registry\...
01d23d9.exe;OPEN;FILE;\c:\windows\system32\shell32.dll;
01d23d9.exe;CREATE;FILE;\device\tcp;
01d23d9.exe;WRITE;REGISTRY;reg_string:\registry\...
                                .
                                .
                                .
01d23d9.exe;CREATE;FILE;\c:\windows\system32\urdvxc.exe;
01d23d9.exe;WRITE;FILE;c:\windows\system32\urdvxc.exe;
01d23d9.exe;OPEN;FILE;\c:\windows\system32\urdvxc.exe;
01d23d9.exe;RELEASE;MUTEX;\shimcachemutex;
```

3.4 Dataset

The dataset used consists of 606 behavior sequences. The dataset was spited into 305 test set and a 301 train set aiming at a 50/50 split of the dataset. In Table 3.2 we can see the distribution of samples for each of the antivirus labels.

Table 3.2: Number of samples for each label.

Test	
ADSPY	10 (3, 28%)
BDS-Hupingon	7 (2, 28%)
BDS-UDR	13 (4, 27%)
DIAL	6 (1, 97%)
GAME-Cassino	3 (0, 98%)
GAME-Dldr-Fenomen	23 (7, 55%)
GAME-Dldr-TryMedia	32 (10, 49%)
TR-Dldr-Swizzor	27 (8, 85%)
TR-Drop	44 (14, 42%)
W32-Virut	24 (7, 87%)
WORM-Allaple	116 (38, 04%)
Total	305 (100, 00%)

Train	
ADSPY	11 (3, 65%)
BDS-Hupingon	10 (3, 32%)
BDS-UDR	17 (5, 65%)
DIAL	12 (3, 98%)
GAME-Cassino	6 (1, 99%)
GAME-Dldr-Fenomen	20 (6, 65%)
GAME-Dldr-TryMedia	29 (9, 64%)
TR-Dldr-Swizzor	20 (6, 65%)
TR-Drop	43 (14, 28%)
W32-Virut	17 (5, 65%)
WORM-Allaple	116 (38, 54%)
Total	301 (100, 00%)

3.5 Clustering

Malware samples from the same family do not have the exact same behavior when executed, one of the ways they can vary is simply in the amount of actions performed. When comparing the lengths of sample sequences for different malware families there can be a big disparity in length as we can see in the Figure 3.1.

As explained previously this large variation on the length of the sequences impacts negatively on the quality of the MSA generated, so in order to improve the PHMM generated we clustered the training samples based on length.

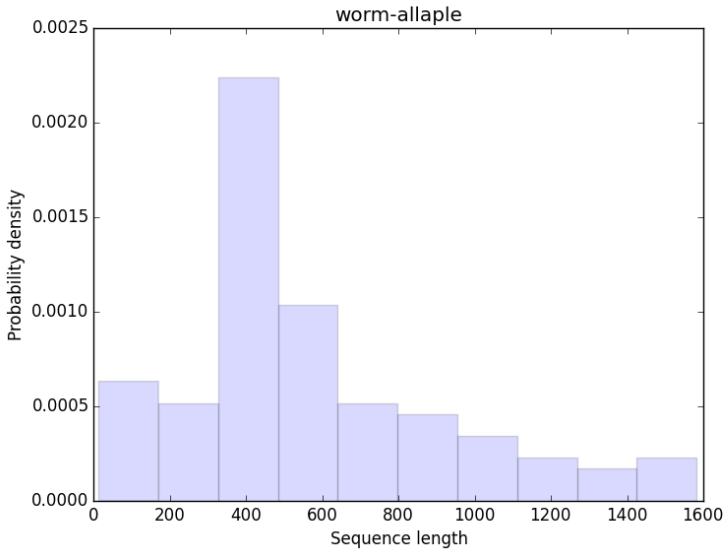


Figure 3.1: Histogram of the length of Worm-Allaple samples.

To do so we used the meanshift algorithm [22] which takes the local maximums obtained from Kernel Density Estimation and we use these as our cluster centers, now for a given label of antivirus we obtain multiple clusters centered around specific lengths of sequences, each of this that will result in a PHHM. Mores graphs of KDE can be seen at Appendix B.

3.6 MUSCLE

The Multiple Sequence Alignments were generated by using MUSCLE [8], which is a public domain software, that performs alignments using a fast distance estimation with k-mer counting. The MSA alignment are stored in the FASTA format

To improve the quality of the alignment we can specify some parameters for MUSCLE, the ones used by us that where different than the default values where: Gap Open cost which specifies the cost for creating a new gap in the alignment, Gap Extend cost which is the cost for increasing the length of a existing gap by 1, and last we have the substitution cost matrix that defines the cost for all possible symbol matching in the alignment. We obtained these parameters through empirical tests, resulting in a Gap Open of -18, Gap Extend of -1 and a substitution matrix as seen in Table 3.3.

Substitution matrices are very specific to the problem and type of alignment being done, and it is a research topic by itself to find optimal values for this matrix in DNA Sequencing [10]. The reasoning behind the values chosen for our matrix is to have the maximum score of 1 for a same symbol match, but also a minor score of 0.3 for matching similar but not same actions, such as READ FILE and WRITE FILE. The encoding of actions was done with the purpose of keeping similar actions in adjacent letters resulting in a better overall appearance to the matrix.

Table 3.3: Substitution matrix used to costs in the sequence aligment

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	1	0.3	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0.3	1	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0.3	0.3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	1	0.3	0.3	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0.3	1	0.3	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0.3	0.3	1	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0.3	0.3	0.3	1	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	1	0.3	0.3	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0.3	1	0.3	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0.3	0.3	1	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	1	0.3	0.3	0.3	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0.3	1	0.3	0.3	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0.3	0.3	1	0.3	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0.3	0.3	0.3	1	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.3	0.3	0.3	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	1	0.3	0.3	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.3	1	0.3	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.3	0.3	1	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

3.7 HMMER

HMMER [13] is an open source implementation of Profile Hidden Markov Models, it comes with several tools and features, those that we used in this thesis follows:

- **hmmbuild** takes a Multiple Sequence Alignment in FASTA format as input and creates a Profile Hidden Markov Model by extracting the probabilities of the model from the alignment.

- **hmmpress** compress various Profile Hidden Markov Model into a proprietary database format that allows efficient and quick to consults.

- **hmmscan** searches sequences against all Profile Hidden Markov Models in a database compressed via *hmmpress*, it outputs an score for each sequence based on the probability of a model generating that sequence, this is score can be filtered by setting a minimum threshold before reporting the result.

3.8 Classifying Malware Sample

To classify a malware sample from our test dataset we use the competing HMM approach used by Kalbhor [14], we run the sample against all PHMM inside the database created by HMMER and assign the label the same as the model with the highest score to the sample.

Chapter 4

Results

In the previous chapter we explained how to obtain Profile Hidden Markov Models, from binary malware samples. In this Chapter we analyze the result obtained by testing the models with a varying threshold parameter.

4.1 Threshold Values

The Threshold Value is the minimum scoring value that a test sample must have in order to consider the results provided by HMMER significant, the scoring value obtained by *hmmscan* is directly related with the probability of Model to randomly generate the sequence being tested, thus a higher score implies in a greater statistical significance.

Figure 4.1: Detection Rate of test samples.

It is important to note that the threshold values impacts on how much of our dataset we are able to assign a label to, test samples where no model scores higher than the

threshold value are considered undecided and thus undetected. In Figure 4.1 we can observe how the detection rates decreases as we increase the threshold value, therefore we would like to have the lowest possible threshold value possible.

However the model metrics are also affected by the decision of a threshold value. In Figure 4.2 we have all evaluation metrics for the BDS-Hupigon and how they change as we vary the threshold value, the more strict we are when making a decision regarding the classification of a test sample the better performance we obtain from our classificatory models.

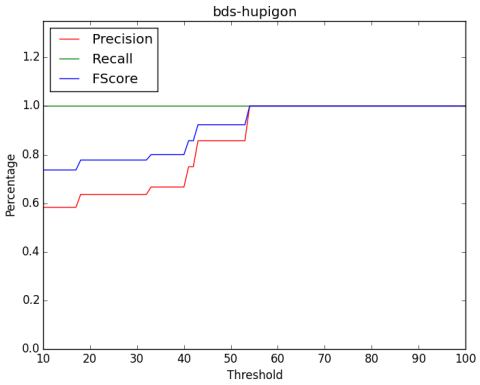


Figure 4.2: Metrics for BDS-Hupingon

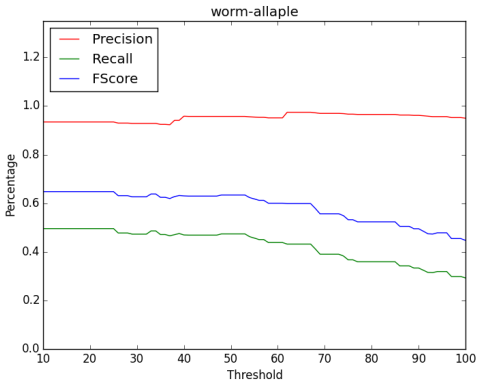


Figure 4.3: Metrics for WORM-Allaple

It is worth mentioning that some models do not improve as we increase the threshold, instead they perform worse, as we can see in Figure 4.3 the graph for WORM-Allaple model. With a manual inspection of the model results, we find that the decreasing of F-Score with increasing thresholds is caused by overall low scores of sequences when matching with the model, this implies that the model too generic being able to only weakly match sequences, and since by increasing the threshold we cut out these matches the model performance decreases causing a lower F-Score. The threshold versus model metrics graph for all other models can be viewed at the Appendix C.

4.2 Model Evaluation

It is possible to define an optimal threshold for each model, with the objective of maximizing both the detection rate as well as the F-Score, but since we have contradicting tendencies there must be a compromise. We chose to give priority to the F-Score, so to find the optimal threshold value we first pick the max possible F-Score and then proceed to search for the lowest possible threshold values that gives us the maximum F-Score, in Table 4.1 we have these optimal values.

Table 4.1: Optimal threshold values and F-score.

Model	F-Score	Precision	Recall	Threshold Value
ADSPY	0.44	0.35	0.6	10
BDS-Hupigon	1.0	1.0	1.0	54
BSD-Udr	1.0	1.0	1.0	26
Dial	0.57	0.44	0.8	30
GAME-Casino	1.0	1.0	1.0	30
GAME-Dldr-Fenomen	1.0	1.0	1.0	22
GAME-Dldr-TryMedia	0.85	1.0	0.75	57
TR-Dldr-Swizzor	1.0	1.0	1.0	49
TR-Drop	0.17	1.0	0.095	28
W32-Virut	0.38	0.24	0.88	48
WORM-Allaple	0.64	0.93	0.49	10

4.3 Confusion Matrix

A confusion matrix is a way to visualize the performance of our models in a multiple classification scenario, each row represents the actual label of the sample and it's distribution of classification across different models, the column represents how a model classified the actual labels. The perfect results would be a confusion matrix where only the diagonals would be dark, meaning that each model only labeled the actual label of the sample.

As previously discussed the threshold values have a direct impact in the performance of the models we can further see that in Figures 4.4 and 4.5 where the result of the matrix improve with higher thresholds as is seen by the darker diagonal.

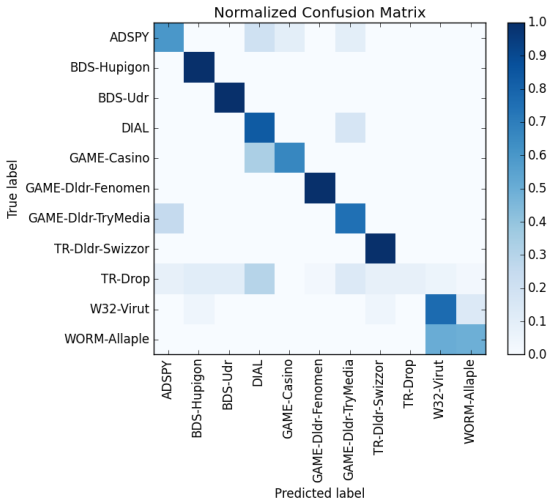


Figure 4.4: Confusion Matrix at 10 Threshold

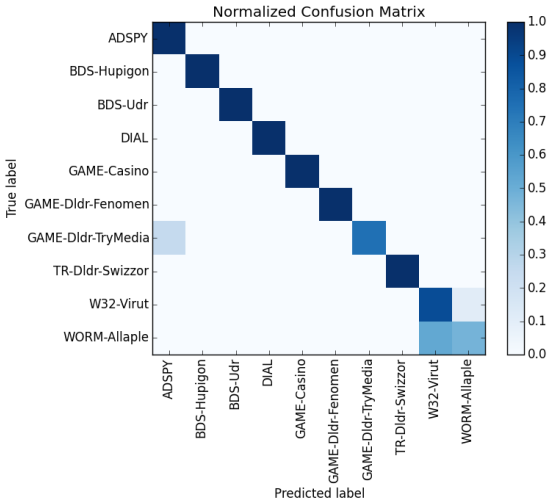


Figure 4.5: Confusion Matrix at 40 Threshold

Another observation is that some models disappear from the confusion matrix when using higher values of threshold such models as seen in Figure 4.6 where the models TR-Drop and TR-Dldr-Swizzor, these models do not appear because they do not provide any results in which the score goes above the threshold, this points that the model is weak and generic and thus not truly representing a label.

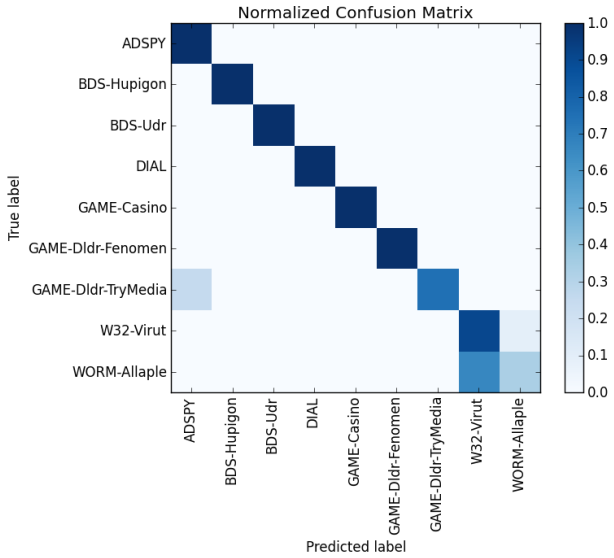


Figure 4.6: Confusion Matrix at 90 Threshold

More confusion matrix plots for different threshold values are available at the Appendix A.

4.4 Allaple and Virut

The matrix also allows us to see models that are similar for example the W32-Virut and WORM-Allaple which consistently where both mismatched at all threshold values, this implies that the mismatch occurs with high scores in the sequences tested.

This does not indicates that the models Virut and Allaple are bad, instead it shows that the labels Virtu and Allaple assignend by the Antivirus (Avira) actually represent a similar malware behavior, since the confusion of the classification only occurs with this two labels and not the others.

4.5 Adspy and Trymedia

The models Adspy and Game-Dldr-Trymedia also shown a small mixture in the confusion matrix for high thresholds, more specifically Trymedia was wrongly classified as Adspy.

Upon further investigation, we found that on some of the Adspy labels give by Avira also contained the terms Trymedia, Casino, and other as we can see in Table 4.2, this points that the samples could have the same behavior pattern but when being detected and classified the antivirus used different methods which results in different labels.

Table 4.2: Adspy labels samples

ADSPY-Trymedia-D_596d93d53b1fba9cb6f48048fe7272625ef4336c.beh
ADSPY-Casino-AA-2_d900f3ce19860a41eade0eaca06028dcb9a60cb2.beh
ADSPY-Downloader-NaviPromo-B-39_9a746868a5a3955bfdee8dc7f3f9c7282f1823df.beh

The difference in classification by the antivirus could be simply a result of an evasion technique from the Malware that evades only parts of the antivirus detection, or it could be explained by the fact that antivirus software is not much worried about correctly classifying malicious programs and more so in determining whether they are malicious or benign with a minimum overhead.

Chapter 5

Conclusion

As discussed in the previous chapter we were able to create sound classificatory models for Malware families based on Profile Hidden Markov Models. There was promising results regarding the ability to statistically model Malware behavior, however there are issues and complications that need to be worked on. In this chapter we explore and discuss some of the limitations found in Section 5.1, followed by suggestions of future work and studies in Section 5.2.

5.1 Limitations

The dataset used originated from the collection of 'wild' Malware in the real world, causing the distribution of Malware families to be skewed towards more common families. This caused some labels in our study to have only a small number of samples which could impact in the performance of the generated models and the reliability of the tests done on those models.

Metamorphic Malwares that do permutation of the order of the actions executed to change its behavior can not be modeled well with Profile Hidden Markov Models simply due to the fact that this kind of transformations is not common in bioinfomatics scenario of DNA Sequencing and thus PHMM states are no designed to model it.

The extraction of behavior logs from malware samples will always be susceptible to the weakness of dynamic analysis, which is anti-visualization techniques used by Malwares to defend themselves against analysis. Another issue originated from dynamic analysis, is that the behavior log sequence of actions may not actually represents the real dependency of the actions due to the nature of multiple process and operating system scheduling, this can impacts the quality of models generated.

Since in this work we used a supervised machine learning approach we are inherently limited by the quality of the original classification provided by the antivirus labels that

where used as grounds truth. Since the main objective of the antivirus is not to correctly classify Malware but separate malicious from benign software in order to protect its user. A solution to this issue would be to find alternative classification of Malwares samples and use those as our ground truth.

5.2 Future Work

The division of the dataset between train and split that was used was a 50/50, this was independent of the number of samples available into the dataset, causing the number of samples used to train each of our models to vary significantly. A study to determine a optimal number of training samples required to generate good models can lead to improvements in the overall performance of these models.

In this thesis we used the simplistic approach of selecting the highest scoring model, however a research of better methods and heuristics to assign labels to test sample based on their score for each model. only, and when manually analyzing the scores obtained by the test samples sometimes the top 5 scores would be very close with the top 1 being and incorrect model and the other 4 the correct label. An interesting approach would be to use a majority council to decide the assigned label in these cases, as well as other methods could improve the overall performance of the models.

An analysis of the states obtained by the creation of the Profile Hidden Markov Models in each of our models can provide insight regarding the underlying behavior of Malware families.

Appendix A

6.1 Confusion Matrices

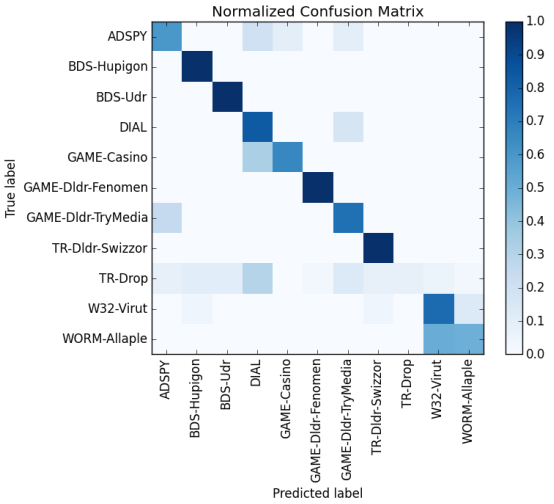


Figure 6.1: CM at 10 Threshold

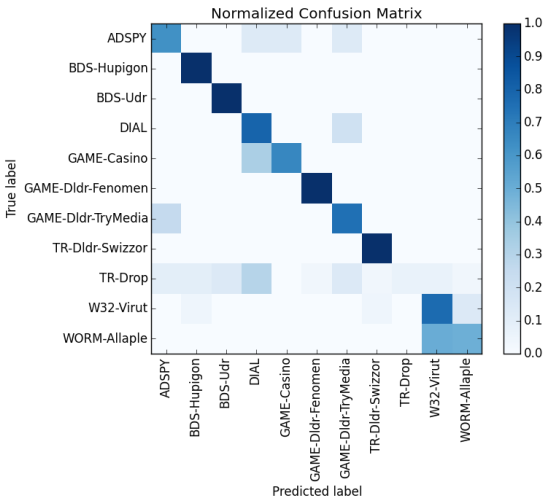


Figure 6.2: CM at 20 Threshold

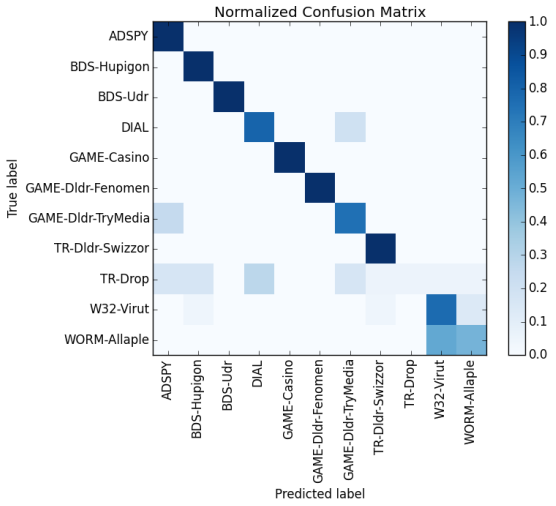


Figure 6.3: CM at 30 Threshold

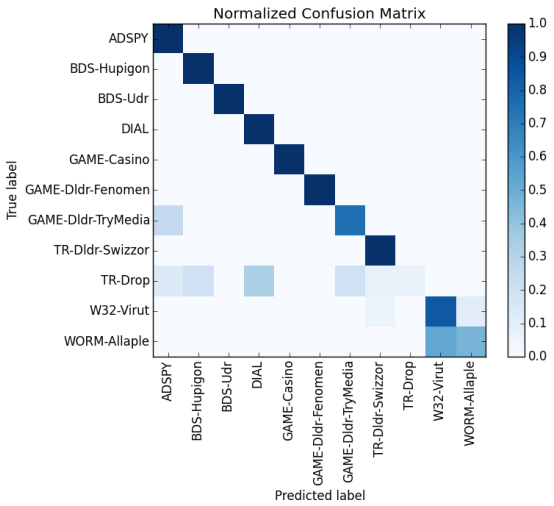


Figure 6.4: CM at 40 Threshold

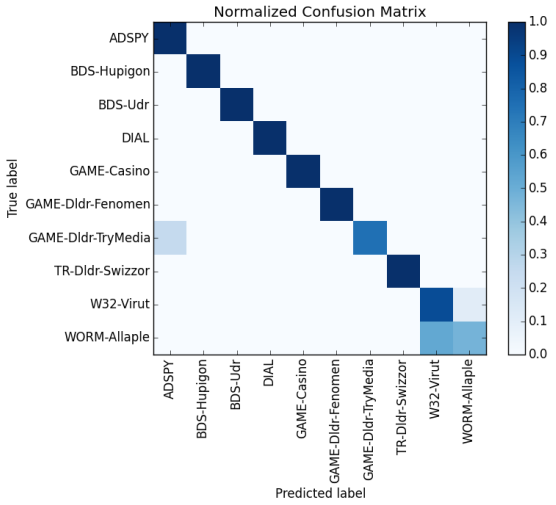


Figure 6.5: CM at 50 Threshold

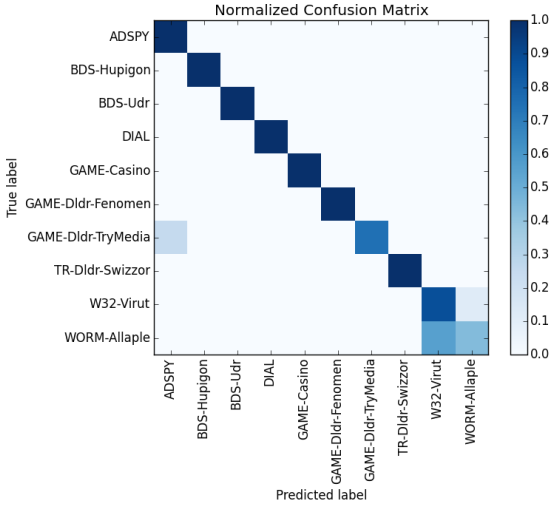


Figure 6.6: CM at 60 Threshold

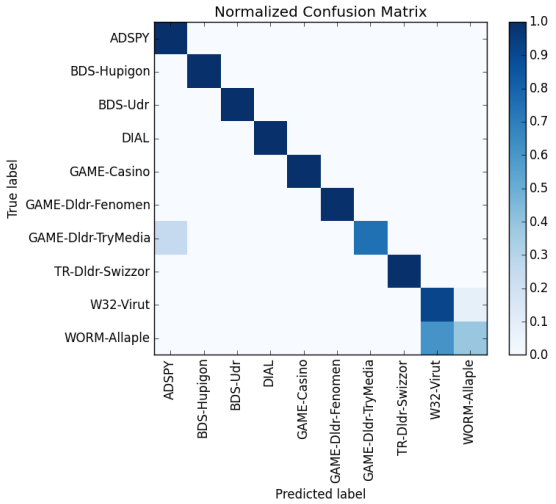


Figure 6.7: CM at 70 Threshold

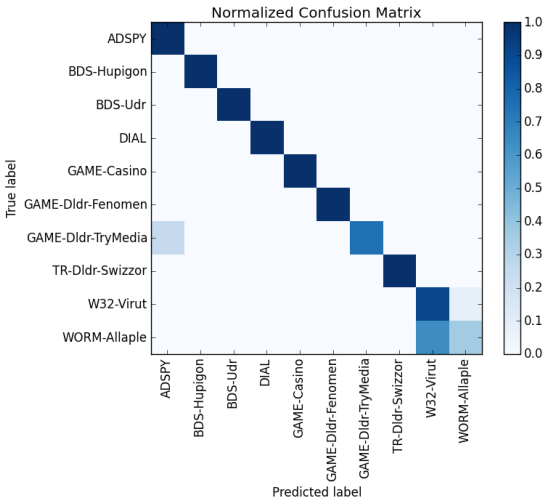


Figure 6.8: CM at 80 Threshold

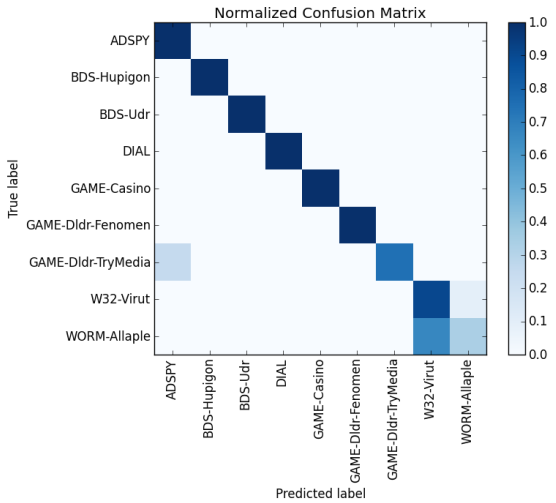


Figure 6.9: CM at 90 Threshold

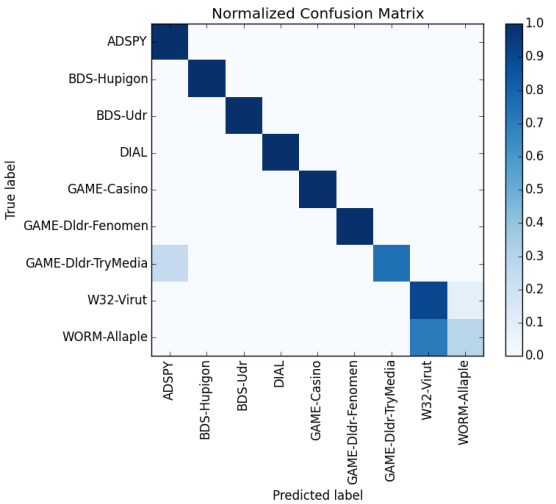


Figure 6.10: CM at 100 Threshold

Appendix B

7.1 Kernel Density Estimation

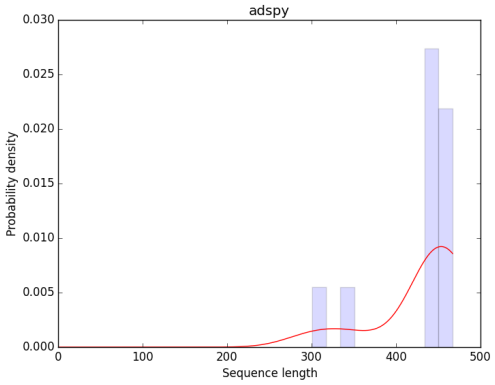


Figure 7.11: KDE for ADSPY

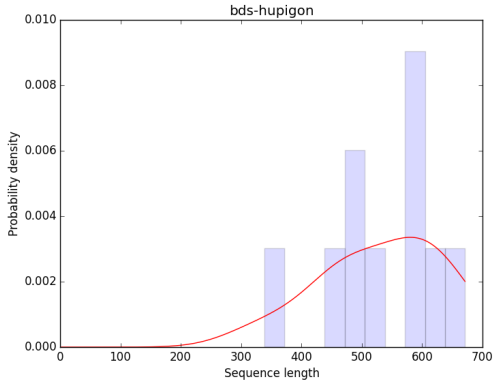


Figure 7.12: KDE for BDS-Hupigon

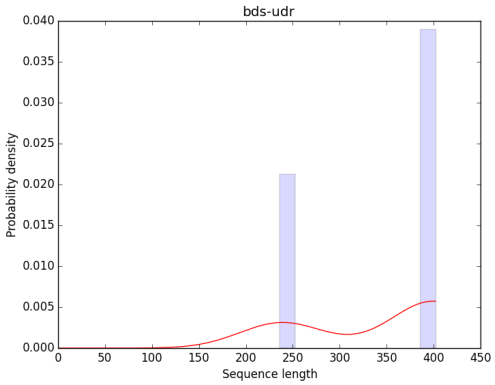


Figure 7.13: KDE for BDS-Udr

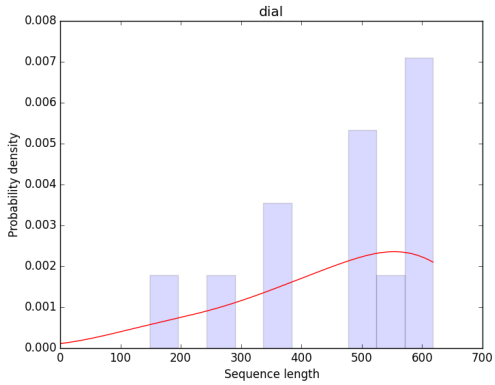


Figure 7.14: KDE for DIAL

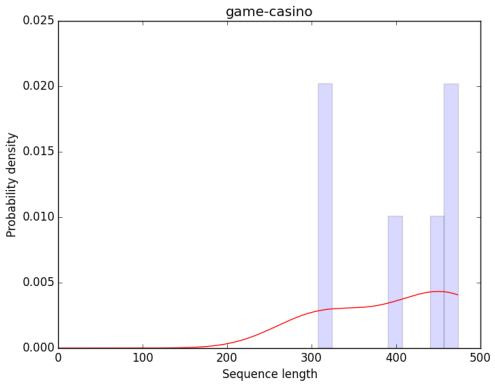


Figure 7.15: KDE for GAME-Casino

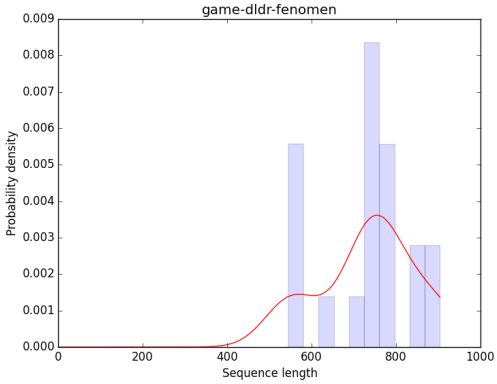


Figure 7.16: KDE for GAME-DLDR-Fenomen

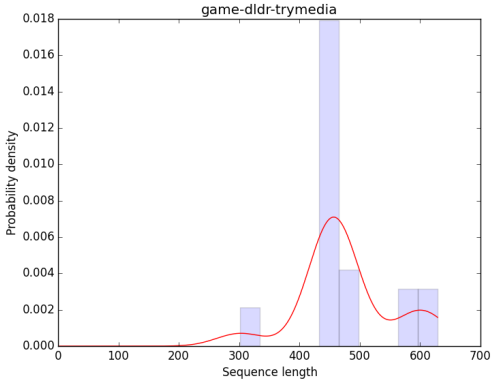


Figure 7.17: KDE for GAME-DLDR-TryMedia

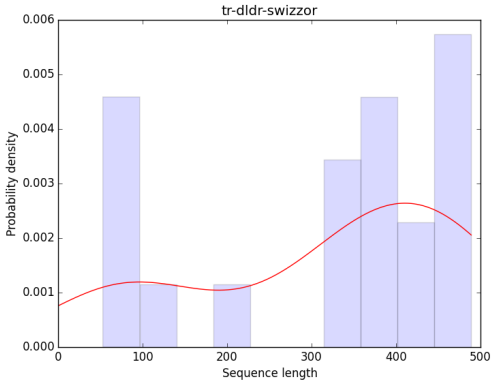


Figure 7.18: KDE for TR-DLRD-Swizzor

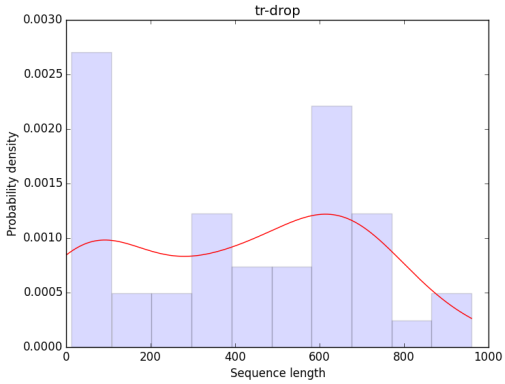


Figure 7.19: KDE for TR-Drop

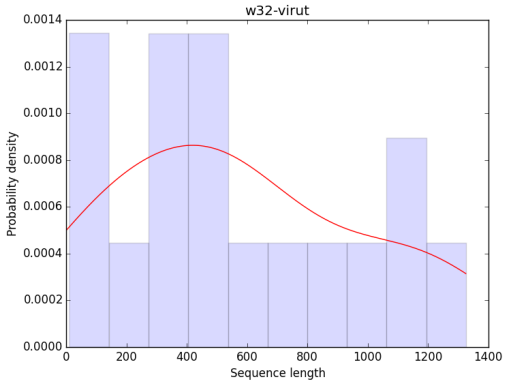


Figure 7.20: KDE for W32-Virut

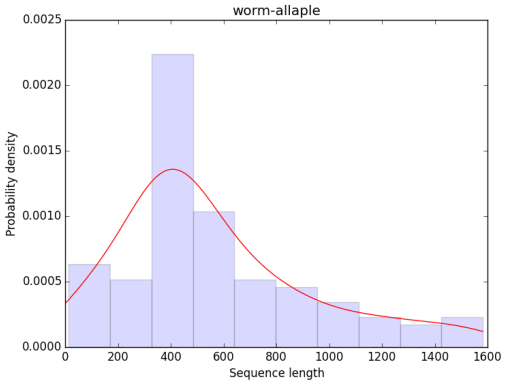


Figure 7.21: KDE for WORM-Allaple

Appendix C

8.1 Model Metrics

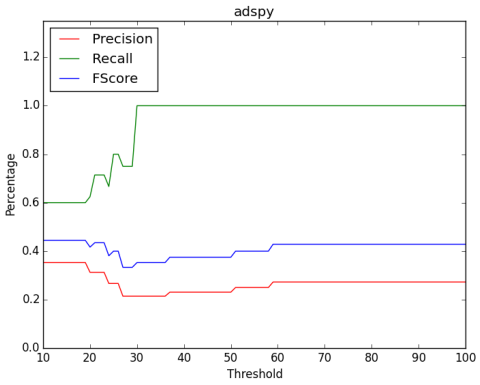


Figure 8.22: Metrics for ADSPY

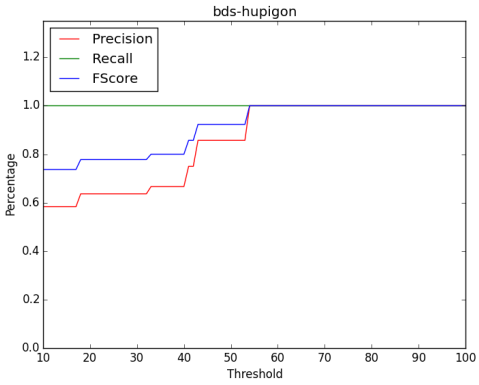


Figure 8.23: Metrics for BDS-Hupigon

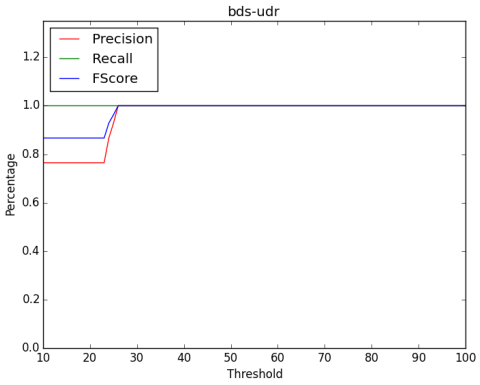


Figure 8.24: Metrics for BDS-Udr

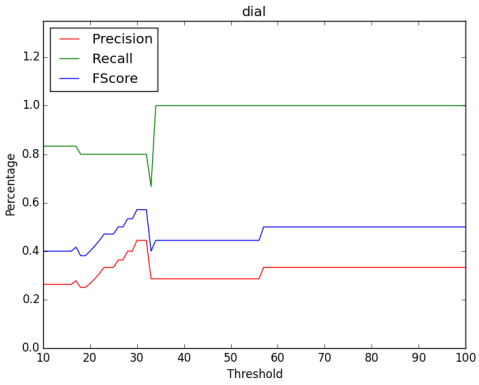


Figure 8.25: Metrics for DIAL

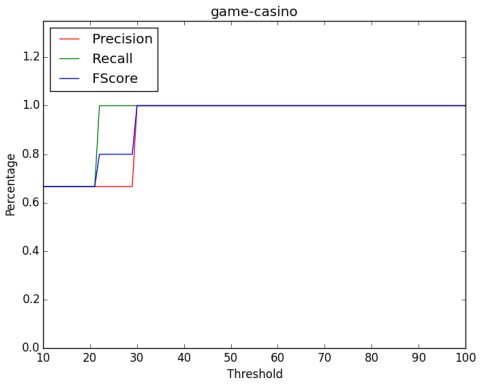


Figure 8.26: Metrics for GAME-Casino

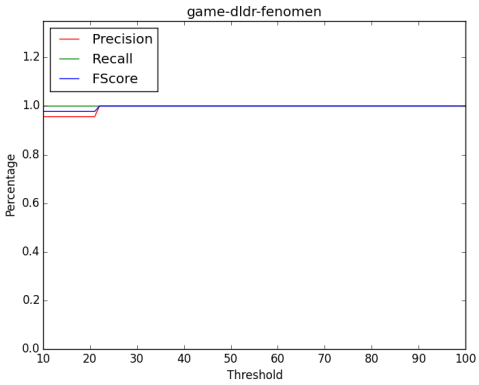


Figure 8.27: Metrics for GAME-DLDR-Fenomen

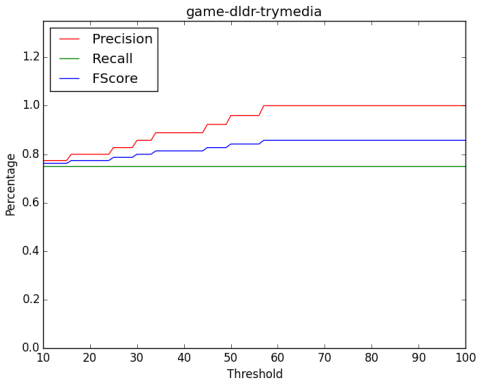


Figure 8.28: Metrics for GAME-DLDR-TryMedia

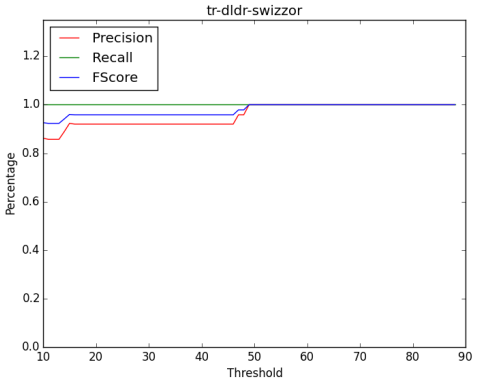


Figure 8.29: Metrics for TR-DLDR-Swizzor

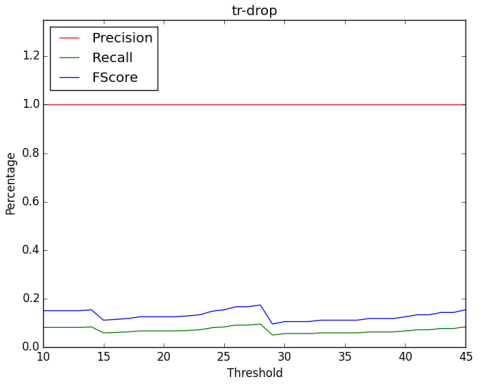


Figure 8.30: Metrics for TR-Drop

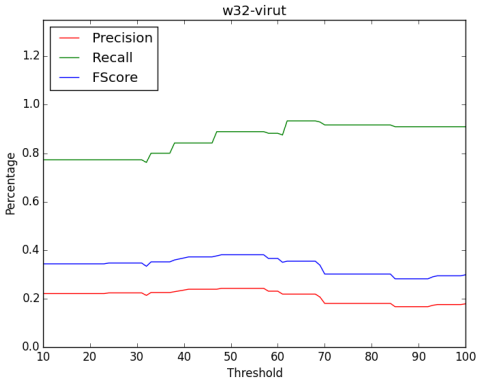


Figure 8.31: Metrics for W32-Virut

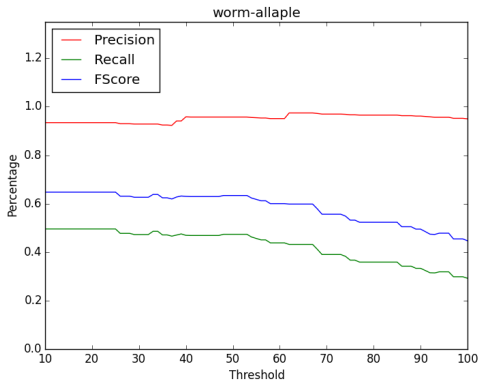


Figure 8.32: Metrics for WORM-Allaple

Bibliography

[1] Chinmayee Annachhatre, Thomas H. Austin, and Mark Stamp. Hidden markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, 11(2):59–73, 2015.

[2] Srilatha Attaluri, Scott McGhee, and Mark Stamp. Profile hidden markov models and metamorphic virus detection. *Journal in computer virology*, 5(2):151–169, 2009.

[3] Thomas H Austin, Eric Filiol, Sebastien Josse, and Mark Stamp. Exploring hidden markov models for virus analysis: a semantic approach. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 5039–5048. IEEE, 2013.

[4] Avira. Antivirus. `https://www.avira.com`.

[5] Virus Bulletin. Reactive and protective tests. `https://www.virusbtn.com/vb100/rap-index.xml`.

[6] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[7] Sean R. Eddy. Profile hidden markov models. *Bioinformatics*, 14(9):755–763, 1998.

[8] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[9] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):6, 2012.

[10] National Center for Biotechnology Information. Blast substitution matrices. `http://www.ncbi.nlm.nih.gov/blast/html/sub_matrix.html`.

[11] G David Forney Jr. The viterbi algorithm: A personal history. *arXiv preprint cs/0504020*, 2005.

[12] André RA Grégio, Dario S Fernandes Filho, Vitor M Afonso, Rafael DC Santos, Mario Jino, and Paulo L de Geus. Behavioral analysis of malicious code through network traffic and system call monitoring. In *SPIE Defense, Security, and Sensing*, pages 80590O–80590O. International Society for Optics and Photonics, 2011.

[13] HMMER. Biosequence analysis using profile hidden markov models. `http://hmmer.org/`.

[14] Ashwin Kalbhor, Thomas H Austin, Eric Filiol, Sébastien Josse, and Mark Stamp. Dueling hidden markov models for virus analysis. *Journal of Computer Virology and Hacking Techniques*, 11(2):103–118, 2014.

[15] Andrew Honig Michael Sikorski. *Practical Malware Analysis*. No Starch Press, 1th edition, 2012.

[16] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pages 421–430. IEEE, 2007.

[17] David W. Mount. *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor Laboratory Press, 2004.

[18] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[19] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[20] R Saradha. Malware analysis using profile hidden markov models and intrusion detection in a stream learning setting. 2014.

[21] Scikit-learn. Kernel density estimation. `http://scikit-learn.org/stable/modules/density.html`.

[22] Scikit-learn. Meanshift. `http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html`.

[23] Mark Stamp. A revealing introduction to hidden markov models, january 2004.

[24] VirusTotal. `https://www.virustotal.com`.

[25] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.