Malware Dynamic Behavior Classification: SVM-HMM applied to Malware API Sequencing

Amanda Stewart ASTEWA35@JHU.EDU
Whiting School of Engineering, Johns Hopkins University
Baltimore, MD 21218

Abstract

Binary classification is more pertinent for today's intrusion detection problem than clustering malware based on families. Detection accuracy carries the greatest risks for anti-virus vendors and security engineers. A high false positive rate, especially in endpoint systems, could result in denial of service for customers. A high false negative rate could result in endpoints being compromised. Performance is another risk where real-time decision-making becomes critical for advanced attacks. Polymorphic malware binaries present additional challenges for static analysis detection systems; as a result, many vendors have pursued virtual emulators for behavioral analysis. A primary focus of this behavioral information research is therefore on classification of sequential events. Markov Chain Models and Hidden Markov Models are most commonly used for modeling anomaly detection and speech recognition through sequential prediction, as well as behavior sequencing. Classification accuracy and concept drift are the primary concerns with these models in the scope of this proposal. I propose that using the Support Vector Machine Hidden Markov Model or SVM-HMM has significant potential advantages that address both of these concerns.

1. Introduction

As Internet technologies evolve and multiply, so does the sophistication and volume of malware. In 2012 there was a 42% increase in targeted cyber attacks where on average 116 to 143 attacks were performed each day (Symantec, 2013). Targeted attacks typically result in the introduction of some of the most sophisticated malware samples at that moment in time. This class of malware introduces malicious features that result in binary polymorphism where each malware sample may be considered unique based on a certain static detection vector. Detection vectors may include hash validation and binary signature detection. Due to the volume and binary uniqueness of malware samples, static detection is becoming less reliable as it does not account for the potential of unknown attacks. It is also difficult to develop signatures based on compressed or obfuscated binary data, since it lacks time sensitivity. Performing dynamic behavior analysis on malware, however, has proven to resolve two of these issues. Similar to the static detection method, signature detection is applied to the identification of malicious dynamic behaviors. The problem with this methodology is that experts develop the signatures so that detection is finite and will not account for potential unknown attacks. Thus there has been an increase in the exploration of dynamic behavior analysis using machine learning. A drawback to this approach is the negative impact to the confidence level in accuracy that is introduced. In malware detection, accuracy is held to the utmost importance to ensure availability and protection to the user.

In my field of experience as a malware research and reverse engineer, I have found that developing signatures for dynamic behavior cannot rely solely on file system and registry modifications in a Windows operating system environment. Using these details alone will result in a sharp uptake of false positive and false negative rates. The threat landscape continually evolves, and

with it so does the sophistication of malware. As evidenced in the profile of targeted attacks, malware focused on stealth may not generate any file system or registry modifications to prevent from being detected by traditional anti-virus scanners. This type of malware would increase the false negative rate if dynamic behavior detection relied on file system and registry changes alone. Instead, monitoring now focuses on a lower monitoring level by intercepting communication between the application and the kernel. This action is also referred to as hooking. Hooking and monitoring for Windows application programming interfaces (API) calls in malicious binaries have shown to be more accurate in describing the underlying behavior.

1.1. Windows API Sequencing

Windows API sequencing is a term used to describe the capture of a sequence of Windows API calls to characterize a certain behavior. Figure 1 depicts an example of a malicious API sequence of the malware labeled by anti-virus signatures as *Trojan.Downloader.Upatre*. In this instance, the malware is creating a new service from a newly created file. It is important to know the placement of this sequence relative to the overall activity of the Windows binary. In the scope of malware detection, signatures or rules are considered generalized when they are created to detect multiple variations of a malware family. Capturing these individual sequences serve as a type of detection generalization where behavior concepts can be applied across multiple windows binaries so that they are able to detect larger clusters of malware.

Seq. No.	API NAME	CATEGORY
60	NtCreateFile	FILESYSTEM
61	NtWriteFile	FILESYSTEM
62	NtWriteFile	FILESYSTEM
63	OpenSCManagerA	SERVICES
64	CreateServiceA	SERVICES
65	StartServiceA	SERVICES

Table 1: Trojan.Downloader.Upatre API sequence excerpt

1.2. Concept Drift

Malware is continuously evolving so that static and dynamic behavior signature generation is reliant upon the malware's characteristics in a given time frame. The term "concept drift" is used in this paper to describe a non-stationary population in the scope of machine learning (Singh, 2012). The malware population can be described as having concept drift because their defining malicious behaviors are not stationary. As such these learned concepts have to change over time through retraining or feature selection. In terms of machine learning, methods of classifying API sequences as malicious means that algorithms need the classifier to adapt to the new concept over an interval of time.

2. Statement of Problem

For machine learning to accurately and effectively classify potential malware samples it must overcome two main obstacles. The first and key one is concept drift. As defined above, concept drift means a constantly changing field that challenges existing learning. In the malware space

those could mean either net new specimens or an old ones that use encryption, polymorphism or code obfuscation. Concept drift is a significant challenge in this field and gets worse as determined adversaries actively seek ways to avoid detection. The second problem is that when traditional algorithms attempt to identify malicious behaviors based solely on individual API events or binary state there is a negative impact on accuracy. In this paper, I propose an approach to overcome those problems based on:

- Improving training data by extracting malware candidate behavioral information with a dynamic behavior analysis system that leverages a virtual machine engine;
- Utilizing SVM-HMM to improve detection accuracy by considering multiple API events in context instead of individual events;
- Overcoming concept drift by shifting the focus from binary static analysis to dynamic behavior analysis. API calls are indeed independent from encryption and other forms of obfuscation.

While concept drift is a significant challenge for machine learning, those key elements should help achieving the goal of detecting malware in a set of candidate samples effectively.

3. Background and Related Work in Field

In recent years there have been few published papers on machine learning concepts with application to API sequencing and there has been even fewer for API sequencing for malware classification. The authors in Rieck et al. (2011), explained one of the first concepts of applying sandbox behavioral output to machine learning. Furthermore, Ravi and Manoharan (2012) explained that malware can be classified by windows API sequencing using a 3rd order Markov chain. Based on their findings, their proposed classifier algorithm using the Markov chain has revealed a better detection rate than that of Support Vector Machines, Decision Tree or Naive Bayes methods.

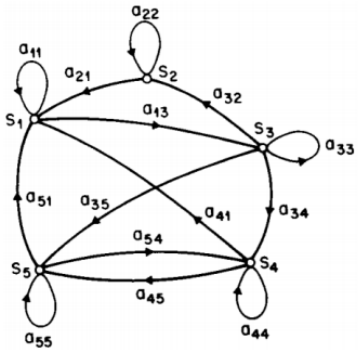

Figure 1 Markov Chain with 5 States (Rabiner, 1989)

3.1. Markov Chain Models

There are several classification algorithms that utilize Markov chains to identify sequences. The key feature of the Markov chain is that it is a stochastic process. Figure 1 represents a simple Markov chain with 5 states where given a set of N=5 states, $S = \{x_1, x_2,\ldots,x_N\}$, a transition be-

tween one state to another is given an associated probability known as a transition probability. The current transition probability is considered independent since it does not rely on future or previous states. Below is a more formal definition of a Markov chain:

Definition 1. Let $X(t) \in S$ indicate the state in time t. Process X(t) is considered a Markov chain where i represents the moment of that state to time, then (Logofet & Lensnaya, 2000)

$$\Pr(X(t_n)=i_n \mid X(t_{n-1})=i_1, \ldots, X(t_1)=i_{n-1}) = \Pr(X(t_n)= i_n \mid X(t_{n-1})= i_{n-1}). \quad (1)$$

A variation of the Markov chain is the Markov chain of order m, where m is finite. In this variation, instead of focusing on 1 prior state, the next state depends on the previous *m* states. This process is defined as the following.

Definition 2. For $n > m$:

$$\Pr(X(t_n)=i_n \mid X(t_{n-1})=i_1, \ldots, X(t_1)=i_{n-1}) = \Pr(X(t_n)=i_n \mid X(t_{n-1})=i_{n-1}, \ldots, X(t_{n-m})=i_{n-m}) \quad (2)$$

In both of these variations, their transition probabilities are denoted as a transition matrix. For each transition matrix *P* has the size for each or *K*x*K* in the order of states. A second order transition probability matrix is shown below (Shamshad, et. al. 2005). The transition probabilities are estimated by the maximum likelihood using the empirical distribution estimated from frequency counts of the transition probability matrices.

$$P = \begin{bmatrix} p_{1,1,1} & p_{1,1,2} & \cdots & p_{1,1,k} \\ p_{1,2,1} & p_{1,2,2} & \cdots & p_{1,2,k} \\ \vdots & \vdots & & \vdots \\ p_{1,k,1} & p_{1,k,2} & \cdots & p_{1,k,k} \\ p_{2,1,1} & p_{2,1,2} & \cdots & p_{2,1,k} \\ p_{2,2,1} & p_{2,2,2} & \cdots & p_{2,2,k} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ p_{k,k,1} & p_{k,k,2} & \cdots & p_{k,k,k} \end{bmatrix}$$

Markov chains by themselves are simple models where the observations are independent of one another. In terms of patterns existing over time, these observations are not generalized. In Ravi and Manoharan (2012), the authors do not address or mention the importance of malware behaviors changing over time. In Singh (2012), the author highlighted the important element of concept drift with malware detection. The author introduced the concept of relative temporal similarity to account for concept drift. This would call for a generalization of the data based on the sequence. Ghaharamani (2001) explained that Hidden Markov Models (HMM) are essentially tools for modeling time series data.

3.2. Hidden Markov Models

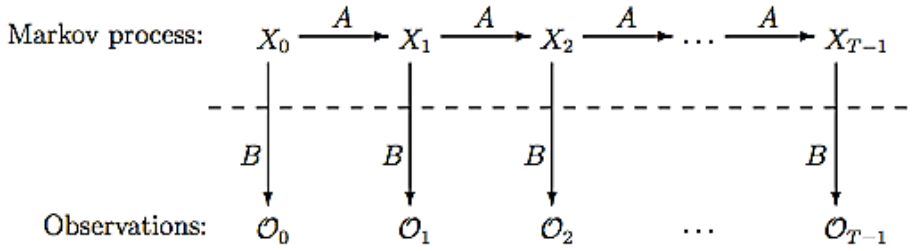

Figure 2: HMM (Stomp, 2012)

Yu (2010) described HMMs as a doubly stochastic process where *s* a discrete-time finite-state homogeneous Markov chain. The hidden element or state sequence influences another stochastic process. In Rabiner (1989), the author describes the most generic definition of an HMM. A HMM is described by the following elements:

1. N as the number of states in the model
2. M as the number of observation symbols per state
3. A as the state transition probability distribution
4. B as the observation probability matrix
5. P_1 as the initial state distribution
6. T as the length of the observation sequence where
7. Q represents the state at that time. These are considered distinct.
8. Given N,M,A,B and the P_1, an HMM can give an observation sequence $O_1,O_2,\ldots,O_n$

Figure 2 demonstrates how these elements are related in the HMM process. In Rabiner (1989), the author also presents three basic problems with HMM that are described in the following questions. For the evaluation problem, given the observation sequence O, and model $m = (A, B, P_1)$, how can we efficiently compute the probability of the observation sequence given the model? For the recognition problem, given the observation sequence O and model $m = (A, B, P_1)$, how do we choose a corresponding state sequence Q that best explains the observations? For the training problem, given the observation sequence O, how would we adjust the model parameters of model $m = (A, B, P_1)$ to maximize probability of the observation sequence given the model?

The solution to the evaluation problem involves enumerating every possible state sequence length. This is computationally expensive in that for every possible N^T state sequence, there are about 2T calculations. This means that there is an order of $2TN^T$ total calculations (Rabiner, 1989). This is where the inference algorithm called the Forward-Backward procedure can help with this computational problem. This algorithm uses dynamic programming principles to compute all of these values efficiently. As for the solution to the recognition problem, the main focus is to choose the states that are most probable and find the optimal state sequence. The Viberti Algorithm is more commonly applied to this problem due to its dynamic programming characteristic.

3.3. Support Vector Machines (SVM) combined with HMM

SVMs have been commonly used for classification (Ravi & Manoharan, 2012). One of the issues with SVMs is maximizing the margin between each class as it may require many calls to separation oracle (Joachims et al., 2009). To address this, a structural SVM framework is proposed by Joachims et al. (2009) paired structural SVMs with a sequence tagging model HMM[1]. Within this structural SVM framework, they provide many parameters for tuning the SVM for any constraints C which is tuning the weight vector, while ε specifies the precision that are required to be satisfied by the solution, and T represents the number of order dependences of the HMM. Joachims et al. (2009) were able to demonstrate that the 1-Slack algorithm dependent on C and ε able to reduce the SVM-HMM training time as well as make training scalable for larger dataset while still under the constraint of the desired precision.

In this design, each input to the HMM where $x = (x_1,\ldots,x_l)$ is a sequence of SVM feature vectors which is one for each event and $y = (y_1,\ldots,y_l)$ are the corresponding sequence of labels. In their SVM framework they are able to utilize the linear chain of HMM to model dependencies between each x and y instance as well as order of dependencies between y instances. The SVM is then able to use a joint feature vector as (1). The loss function is defined by the number of misclassified labels as $\Delta((y_1,\ldots,y_l),(y_1,\ldots,y_l)) = \Sigma_{li=1}[y_i \_= y_i]$. In their framework they have chosen the Viberti algorithm to compute the *argmax* for prediction and the separation oracle. Although this design was originally meant for Part-of-Speech Tagging, it is applicable to tagging the benign and malicious states of API call sequences and has a favorable training time. In this paper, the SVM-HMM framework designed by Joachims et al. (2009) will be integrated into the proposed malware analysis system.

$$\Psi HMM((x1,\ldots,xl),(y1,\ldots,yl)) = \sum_{i=1}^{l} \begin{pmatrix} \Psi multi\ (xi,yi) \\ [yi=1]\,[yi-1=1] \\ [yi=1]\,[yi-1=2] \\ \vdots \\ [yi=k][yi-1=k] \end{pmatrix} \quad (1)$$

4. Proposed Methodology

The data collection process will entail gathering a mixture of benign and malicious windows executables from a public and community driven malware analysis repository known as VirusTotal.com[2]. Expert assigned malware sample criteria requires a minimum of 20 antivirus vendors declaring the sample as malicious. Expert assigned benign samples have been positively identified as being a legitimate executable, meaning that no antivirus vendor should have declared the sample as malicious. The training dataset will include 2000 benign and 953 malicious samples. This dataset will be submitted to an online sandbox analysis website called malwr.com[3]. Malwr.com uses that sandbox to report on the dynamic process information from that executable. Figure 3 shows the high level system architecture and information flow. Table 2 is an example of

[1] SVM-HMM framework. http://www.cs.cornell.edu/people/tj/svm_light/svm_hmm.html

[2] "VirusTotal, a subsidiary of Google, is a free online service that analyzes files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners." https://www.virustotal.com/en/about.

[3] "Malwr is a free malware analysis service and community launched in January 2011." http://malwr.com/about/

malwr.com's behavior analysis report output and will serve in producing the main features of the dataset. The sequence number will serve in tracking the position in sequence and the sequential feature in the SVM-HMM. The API name and category will serve as a secondary classification feature vector for the SVM.

UNIQUE ID	SEQ	API NAME	CATEGORY	Label
7e55f90729ff5b2539d11f55453bb7b9	60	NtCreateFile	FILESYSTEM	Malicious
7e55f90729ff5b2539d11f55453bb7b9	61	NtWriteFile	FILESYSTEM	Malicious
7e55f90729ff5b2539d11f55453bb7b9	62	NtWriteFile	FILESYSTEM	Malicious
7e55f90729ff5b2539d11f55453bb7b9	63	OpenSCManagerA	SERVICES	Malicious
7e55f90729ff5b2539d11f55453bb7b9	64	CreateServiceA	SERVICES	Malicious
7e55f90729ff5b2539d11f55453bb7b9	65	StartServiceA	SERVICES	Malicious

Table 2: Data Set Features

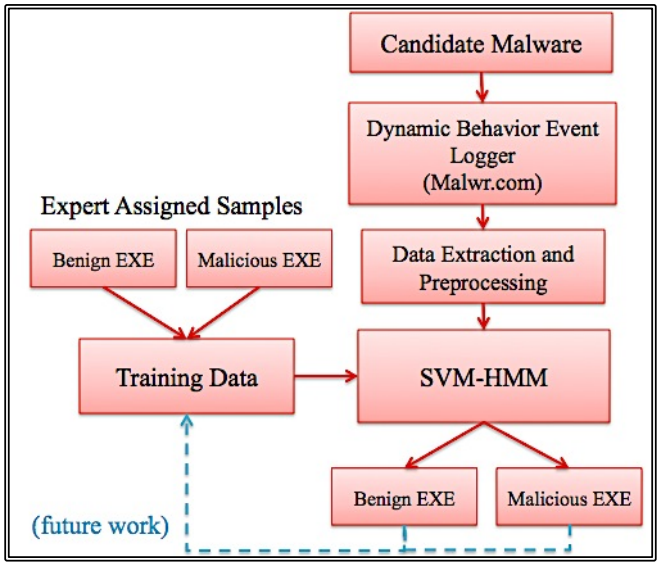

Figure 3: Proposed System

Malwr_Learn(training_candidates, parmeters: *C Constraint, ε Espsilon, T Order, Emission)*

```
Let DB be the API mapping database
Let e ∈ E events of each candidate where e[category, apicall]
Traning_Data
for each candidate ca ∈ Ca candidates:
    for each expert_label of e and e of c:
        m = Map(e, DB) // Translate each category and API call event to integer representation
        Training_Data.add(expert_label, m)
Learned_model = SVM-HMM.learn(Training_Data, parameters)
Return Learned  model
```

Figure 4: Malwr Learning Algorithm

```
Malwr_Classifier(candidates, threshold)
   Let DB be the API mapping database
   Let e ∈ E events of each candidate where e[category, apicall]
   for each candidate ca  ∈ Ca candidates:
       Candidate_sequence
       for each e of ca:
            m = Map(e, DB) // Translate each category and API call event to integer representation
            predicted_state = SVM-HMM.classify(m) // benign or malicious
            Candidate_Sequence.add(predicted_state)
       ratio = Candidate_Sequence with malicious state/Size of Candidate Sequence
        if ratio >= threshold:
            return malicious
         else:
             return benign
```

Figure 5: Malwr Classification Algorithm

Figures 4 and 5 represent the two main algorithms of the proposed system[4]. Both algorithms use the SVM-HMM framework proposed by Joachims et al. (2009). The nuance about the classification algorithm is that once all labels are given to the predicted values, the ratio is calculated over all events in the candidate runtime sequence. If the ratio is above the given threshold, it will be marked as anomalous.

4.1. Experimental Design

Table 3 contains the distribution of malware families which (1) have been known to exhibit concept drift (Singh, 2012) and (2) have been known to exhibit polymorphism characteristics in the malicious training dataset. Each family contains several variants that have appeared over several months. For each test case the malware variants will be tested with 10-fold cross validation to test all three families that exhibit concept drift. Figures 6 and 7 represent the current API call event patterns for each separate dataset. These graphs were extracted from the candidate reports and mapped using the API mapping function map(). Each sample in the data set is overlaid in the graph, the darker coloring show the most common API call patterns that exist across the dataset. There are many observable commonalities between the 2 data sets where the first ~5 events have similar API call patterns. On the other hand, the events at index 15-20 begin to visually show contrast between the two data sets.

NO. DIFFERENT VARIANTS	FAMILY NAME	NO. OF EXAMPLES
132	Agent	189
11	Uptare	238
101	Zbot	526

Table 3: Malware Families

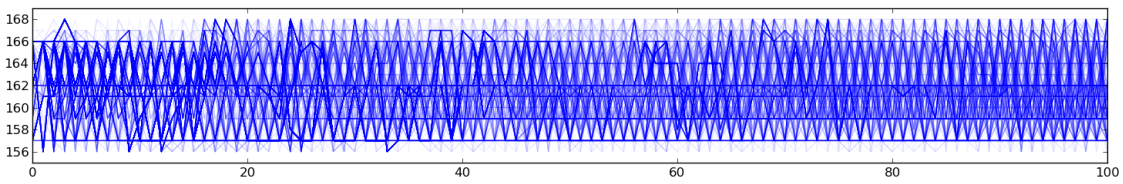

Figure 6: 2000 Benign Executables for 100 events

[4] Source code and datasets can be found here: https://github.com/astewa/MF

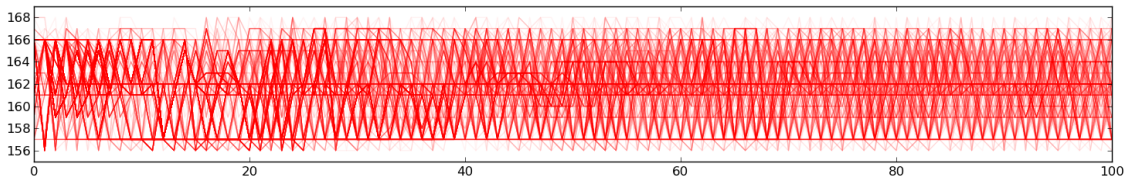

Figure 7: 953 Malicious Executables for 100 events

The proposed system will be compared with a previous API call classification system developed by Ravi and Manoharan (2012). Figure 8 represents the interpreted algorithm for the classifier and placement of the *rule minor* algorithm (Ravi & Manoharan, 2012). This system uses two variables support and confidence to satisfy the loss function. These variables are defined below.

1. I = candidate item
2. Count (I U {Class}, DB) is the no. of records in the DB in which (I U {Class}) holds
3. Class represents malware or benign
4. DB is the signature database
5. |DB| is the no. of records in DB
6. Support = [Count (I U {Class}, DB) / |DB|] x 100%
7. Confidence = [Count (I U {Class}, DB) / Count (I, DB)] x 100%

```
Ravi_Classifier(candidates)
   Let RDB be the rules database
   Let DB be the API mapping database
   Let e ∈ E events of each candidate where e[apicall]
   for each candidate c  ∈ C candidates:
      Candidate_sequence
      for each e of c:
           m = Map(e, DB) // Translate each category and event to integer representation
           support[malware class, benign class],
           confidence[malware class , benign class] = rule_miner(m, RDB)
           if support >= threshold:
                RDB.add(m, class_label)
           Candidate_Sequence.add(confidence)
      if Average (Candidate_Sequence[confidence[malicious]) > Average(Candidate Se-
quence[confidence[benign]]):
           return malicious
       else:
            return benign
```

Figure 8: Ravi Classifier Algorithm

As for empirically evaluating the two binary classification systems, the experimental results give the classifier accuracy. Accuracy is derived from the confusion matrix (Figure 9) where each predicted positive and predicted negative is compared to the true positive and true negative labels (Powers, 2007). Accuracy is then defined by (2). Figure 9 describes the algorithm chosen for calculating the average area under the curve (AUC) from a Receiver Operating Characteristics (ROC) graph which are a useful technique for organizing classifers and visualizing their performance. Among the 10-fold cross validation, the ROC graph and AUC will be used for selecting the optimum threshold for the proposed system. The Ravi and Manoharan (2012) system does not utilize a classification threshold.

	Predicted Positive	Predicted Negative
Actual Positive	TP = True Positive	FP = False Positive
Actual Negative	FN = False Negative	TN = True Negative

Figure 9: Confusion Matrix

$$\text{Accuracy} = ((TP+TN) /(FN+TN+TP+FP)) \quad (2)$$

Calculating the area under an ROC curve

Let L be the set of test examples; f(*i*), the probabilistic classifier's estimate that example *i* is positive; P and N, the number of positive and negative examples.
Outputs: A, the area under the ROC curve.
Require: P > 0 and N > 0

```
Lsorted ←L sorted decreasing by f scores
FP ←TP← 0
FPprev ←TPprev ←0
A←0
fprev ←-∞
i← 1
while i ≤ |Lsorted|  do
   if f(i) 6= fprev  then
      A ←A + trapezoid area(FP; FPprev ; TP; TPprev)
      fprev← f(i)
      FPprev← FP
      TPprev ←TP
   end if
   if i is a positive example  then
      TP ←TP + 1
   else
      FP ←FP + 1
   end if
end while
A ←A + trap area(1; FPprev; 1; TPprev)
A ←A=(P _ N) /* scale from P _ N onto the unit square */
end

function trapezoid area(X1;X2; Y 1; Y 2)
   Base ←|X1 - X2|
   Heightavg ← (Y 1 + Y 2)=2
   return Base _ Heightavg
end function
```

Figure 10: AUC Algorithm (Fawcett, 2004)

5. Experimental Results

In this section, Figures 11-13 represents the ROC curves for each 10-fold cross validation set to each chosen *T* order of dependences. Parameters for choosing C=10, and ε=0.001 were chosen for each test case for *T*. Table 6 represents the Accuracy rate and AUC rate for each T test case.

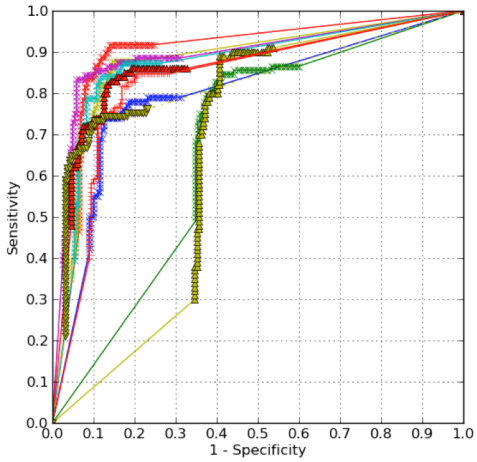

Figure 11: 10-fold cross validation of C=10, Epsilon = 0.001, T Ordered Dependent States=1, Emission = 1

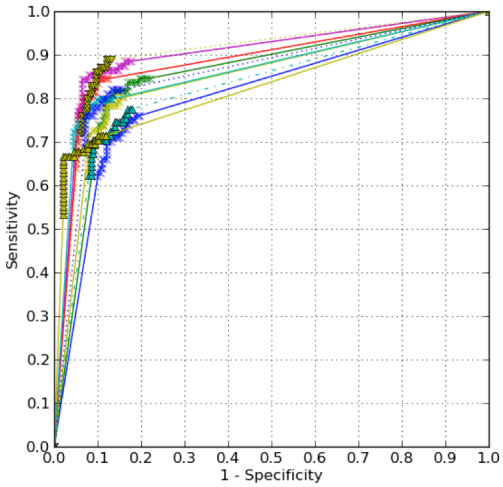

Figure 12: 10-fold cross validation of C=10, Epsilon=0.003, T=1,E=1

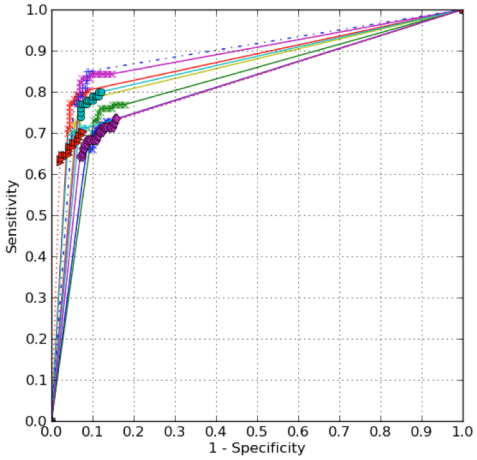

Figure 13: 10-fold cross validation of C=10, Epsilon=0.001, T=3,E=1

AVG AUC 10 FOLD	STANDARD ERROR	T ORDER DEPENDENCIES	EMISSION	EPSILON SVM	CONSTRAINT S C	AVG ACCURACY
0.818339793	0.02697486	1	1	0.001	10	0.827090301003
0.857098284	0.02518974	2	1	0.001	10	0.867558528428
0.846071979	0.02606259	3	1	0.001	10	0.867892976589

Table 4: SVM-HMM Parameter Selection

SYSTEM	TEST AVG ACCURACY	Learning + Classification Runtime
Dynamic Behavior Classifier using SVM-HMM 3 Order	0.867892976	11.05 seconds
Ravi Model 4-gram	0.367558528	4130.44 seconds

Table 5: Comparison Between Ravi's Proposed System and SVM-HMM System

6. Discussion and Analysis of Results

The experimental results for the proposed system *T* ordered dependencies case study and threshold selection revealed that *T*=3 results in the strongest and most stable accuracy rate across the 10-fold cross validation. The consistency in Figure 11 results revealed that the training set contained several Zbot malware variants in which had many benign behavior characteristics. In my experience, Zbot is a very elusive malware that exhibits very few dynamic behavior events but has one of the highest polymorphic characteristics among the three sets of malware chose for this study. Figure 12 and 13 results show that utilizing the HMM order dependency parameter of the SVM-HMM classifier exhibited a more consistent ROC curve over each of the 10 folds.

In Table 7, we observe that the accuracy rate of the Ravi Model 4-gram had a very poor accuracy for classifying the malware variants. With accuracy being the main focus of this paper it is also interesting to note the approximate average performance time for learning and classification combined. The Ravi system had a classification and runtime took longer than an hour to classify each sample. This may be due to the constraint of search algorithm for searching the rule database. Ravi and Manoharan (2012) did not mention the best optimization for the *rule mining* algorithm. Despite the runtime metric, the Ravi and Manoharan (2012) model produced a very low accuracy rate. The malware detection industry has high standards for detection rates. Anything with below 95% accuracy would not be considered as a potential detection engine. The initial proposed system did achieve 86.76% accuracy rate. Since only *T* order dependencies were explored for this case study there are other parameter tuning case studies that can stem from this study.

7. Relevant and Future Work

In conclusion, the SVM-HMM is a promising approach for classifying dynamic behavior and current accuracy could be improved upon with additional research. There are still many other possible test cases for parameter selection. Improving other aspects of the proposed dynamic behavior detection engine could be done, as mentioned in the architecture diagram, by reusing high-confidence output data. This is similar to improvement approach in the detection engine proposed by Ravi and Manoharan (2012). While the three tested malware families have exhibited concept drift in the past, it remains to be seen how similar would the characteristics to other families. Another possible approach to this would be to create a separate SVM-HMM for each malware family type (ex: downloaders, rootkit, etc.) because they represent drastically different behaviors sets. There also can be exploration into adding support for API call arguments however it would add challenging complexity to the learning algorithm. In this paper, the focus of the analysis was on the call itself and not its arguments. Figure 14 shows an example of an API call that contain up to 3 or more arguments. These arguments could contain a wide range string values that may not be applicable for a SVM.

In the malware detection industry, purely detecting anomalous behavior is favored over malware family classification. Xie et al. (2010) presented another usage of HMM for anomaly detection but the hidden state is based on a small range of predicted states and a rule database.

TIME	APICALL	ARGUMENTS	STATUS	RETURN
014-04-25 04:06:32,153	NtOpenSection	DesiredAccess:0x000f001f ObjectAttributes:C:\ntdll SectionHandle:0x000000a4	success	0x00000000

Figure 14: API call function parameters

The usage of Hidden Semi-Markov Models may be useful for describing anomalous sequencing in executables. Yu (2010) explains that the Hidden Semi-Markov model (HSMM) is able to overcome many of the limitations of HMM. In (Oura et al. 2006), the authors successfully apply HSMM to speech recognition and produce more accurate results than HMM. Seeing that HSMM produces more accurate results, this may be a viable candidate for API sequencing analysis.

References

Rieck, K., Trinius P., Willems, C., Holz, T. (2011) Automatic Analysis of Malware Behavior using Machine Learning. Journal of Computer Security. IOS Press.

Ravi, C., Manoharan, R. (2012). Malware Detection using Windows API Sequence and Machine Learning. International Journal of Computer Applications (0975 – 8887) Volume 43– No.17.

Singh, A. Walenstein, A., Lakhotia, A. (2012). Tracking Concept Drift in Malware Families. Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence (AISec'12).

Ghaharamani, Z. (2001). An Introduction to Hidden Markov Models and Baysien Networks. International Journal of Pattern Recognition and Artificial Intelligence 15(1):9-42.

Baum, L., Petrie, T., Soules G., Weiss, A. (1970). A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains, Annals of Mathematical Statistics, Vol. 41, No. 1, 1970, pp. 164–171.

Welch, L. (2003). Hidden Markov Models and the Baum-Welch Algorithm, IEEE Information Theory Society Newsletter.

Singh, A. (2012). On Concept Drift, Deployability, and Adversarial Selection in Machine Learning- Based Malware Detection. University of Louisiana at Lafayette.

Sahoo, N., Singh, P., Mukhopadhyay, T. (2012). A Hidden Markov Model for Collaborative Filtering. MIS Quarterly.

Yu, S.-Z. (2010). Hidden semi-Markov models. Artificial Intelligence 174, pp. 215–243.

Oura, K., Zen, H., Nankaku, Y., Lee, A., and Tokuda, K. (2006). Hidden Semi-Markov Model Based Speech Recognition System Using Weighted Finite-State Transducer. Nagoya Institute of Technology Department of Computer Science and Engineering, Gokiso-cho, Showa-ku, Nagoya, Japan.

Powers, D. (2007). Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation No. SIE-07-001.

Rabiner, L. R. (1989).A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE, Vol. 77, No. 2.

Stamp. M. (2012). A Revealing Introduction to Hidden Markov Models. Department of Computer Science San Jose State University.

Yi, X., Shunzheng, Y. (2005). A Detection Approach of User Behaviors Based on HsMM*. Department of Electrical and Communication Engineering Sun Yat-Sen University, Guangzhou 510275, China.

Logofet D.O, Lensnaya E.V. (2000). The mathematics of Markov models: what Markov chains can really predict in forest successions. Ecological Modelling 2000; 2-3:285-298

Symantec Corporation. (2013). Internet Security Threat Report. 2012 Trends, Volume18, Published April 2013

Shamshad, A. Bawadi, M. A., Wan Hussin, W. M. A., Majid, T. A., Sanusi, S. A. M. (2005). First and Second Order Markov Chain Models for Synthetic Generation of Wind Speed Time Series. Energy, vol. 30, pp. 693-708.

Johnson, M., Willsky, A. (2013). Bayesian Nonparametric Hidden Semi-Markov Models. Journal of Machine Learning Research 14 (2013) 673-701.

Yi, X., Shunzheng, Y. (2009). A Large-Scale Hidden Semi-Markov Model for Anomaly Detection on User Browsing Behaviors. IEEE/ACM Transitions on Networking, VOL. 17, NO. 1.

Xie, L., Seifert, J., Zhang, X., Zhu, A. (2010). pBMDS: A Behavior-based Malware Detection System for Cellphone Devices. WiSec'10 Proceedings of the third ACM conference on Wireless network security, pp. 37-48.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. International Joint Conference on Artificial Intelligence.

Joachims, T., Finley, T., and Yu, C. (2009). Cutting-Plane Training of Structural SVMs, Machine Learning Journal, 77(1):27-59.

Fawcett, T. (2004). ROC Graphs: Notes and Practical Considerations for Researchers. HP Laboratories.