Early Stage Malware Prediction Using Recurrent Neural Networks

Matilda Rhode*, Pete Burnap*, Kevin Jones†,
*Cardiff University
†Airbus Group

Abstract—Certain malware variants, such as ransomware, highlight the importance of detecting malware prior to the execution of the malicious payload. Static code analysis can be vulnerable to obfuscation techniques. Behavioural data collected during file execution is more difficult to obfuscate, but typically takes a long time to capture.

In this paper we investigate the possibility of predicting whether or not an executable is malicious. We use sequential dynamic data and find that an ensemble of recurrent neural networks is able to predict whether an executable is malicious or benign within the first 4 seconds of execution with 93% accuracy. This is the first time a file has been predicted to be malicious during its execution rather than using the complete log file post-execution.

I. INTRODUCTION

The aftermath of successfully executed malware attacks can be costly, whether this entails network repair or data recovery. Ransomware attacks, for example, typically demand financial compensation in exchange for a decryption key to unlock data, and have seen a 300% rise in 2016 compared with 2015 [1]. Such attacks may also incur the less-quantifiable cost of user trust and/or organisation reputation due to service disruption. In this paper, we investigate the possibility of predicting whether or not a file is malicious based on the initial behaviour of the executable. The motivation for this research is to detect malware before the malicious payload is executed, thus using behavioural data to prevent damage rather than a means to understand the repairs necessary after the damage has been done. As far as we are aware, this is the first paper attempting to predict whether or not a file is malicious using initial behavioural activity. To date, dynamic malware detection research has made use of entire file execution logs, or simply chosen some fixed time at which to stop recording. Using sequential machine activity data, we explore the predictive capabilities of recurrent neural networks to determine whether or not a file is malicious.

Malware detection today needs to be automated due to the increasing volume of files submitted for analysis. The free file analysis tool, VirusTotal, can receive more than one million distinct files for investigation in a single day [2]. Signature-based detection, which identifies malware by comparing known signatures to file contents, can be evaded by obfuscating code. Signature-based systems are particularly unsuited to detecting zero-day malware unless it shares signatures with previously known strains [3]. Conversely, anomaly-based systems compare a baseline of “normal” activity and capture deviations from this baseline. Anomaly detection creates an alert for any aberrant behaviour rather than for recognised signatures. Deviations from baseline activity must then be investigated, by which time a malicious executable may have already achieved its aims. Machine learning seeks to remedy the tendency of signature-based systems to falsely classify malware as benign and the tendency of anomaly-based systems to falsely classify benign behaviour as a risk.

The data used by machine learning algorithms to classify executables may be static, dynamic or a combination of the two. Static data is scraped directly from source code, and can be used to classify files before execution. Static data, however, can be manipulated to evade accurate classification [4]. Dynamic data is recorded during file execution in a virtual machine and seeks to capture the actual processes carried out by the file. It is more difficult to mask malicious activities when data is collected in this way, as malware must at least carry out the processes necessary to realise its objectives.

Existing research using machine learning and dynamic data to detect malware uses the entire execution cycle e.g. (5, 6, 7) or choose some arbitrary time at which to cap recording e.g. (8, 9, 10). Some approaches examine the length of time for which to record data based on file properties e.g. (11, 12). Such approaches focus on reducing dynamic data recording time on an instance-by-instance basis, meaning that full dynamic analysis is still carried out for some samples.

Our approach will primarily use the initial seconds of file execution. This reduces the amount of information available to the classifier. Owing to the sequential nature of dynamic data, we believe that a classifier capable of understanding the relationships between features over time will capture more information about each sample and thus be capable of more accurate classification. Recurrent neural networks (RNNs) are able to process time series data [13] and we show that RNNs outperform other widely-used machine learning algorithms in predicting malware.

The key contributions of this paper are:

- Demonstration that malware can be predicted from the first 4 seconds of file execution with 93% classification accuracy.
accuracy.

- Analysis of classification accuracy as time into file execution progresses.
- We show that it is possible to reduce the time spent recording sample data by 98% (4.8 minutes per sample to 4 seconds per sample) and achieve an accuracy of 93%, while 98% accuracy can be achieved using the first 20 seconds of data (93% time reduction).

II. RELATED WORK

A. Malware Classification

Malware detection is a complex classification problem partly because adversaries adapt malware to avoid detection. The resilience of machine learning-based detection can depend on the kind of data used. Static data, collected directly from code, is quick to collect and can be analysed prior to file execution. Saxe and Berlin [14] distinguish malware from benignware with an accuracy of 97.45% using a deep feed-forward neural network trained on static program features. Grosse et al. [12] demonstrate, however, that static data can be obfuscated to cause a classifier previously achieving 97% accuracy to fall as low as 20% when classifying adversarially crafted samples, with some accuracy increase possible using defensive techniques such as training on obfuscated samples. This vulnerability to obfuscation renders static models less reliable than dynamic models [15].

Dynamic runtime behaviours such as system calls to the operating system kernel are the most common features used in dynamic analysis. Beyond resilience to obfuscation, dynamic data has been found to give better classification accuracy. Damodoran et al. [16] compare malware classification of static and dynamic models using Hidden Markov Models (HMMs) and find that dynamic data yields the highest accuracy with a 0.98 Area Under the Curve score. Huang and Stokes [5] were able to classify malicious and benign executables with an accuracy of 99.64% from 114 System API calls and features derived from those API calls using a shallow feed-forward neural network.

Dynamic data, though more robust to obfuscation, takes much longer to collect and so is impractical (by comparison with static analysis) for analysing files in real-time as part of an endpoint defensive system. Waiting for the entire file to execute in a virtual machine or emulator before classifying creates an impractical delay if used as part of an endpoint security system.

B. Reducing dynamic data recording time

Existing methods to reduce dynamic data recording time focus on efficiency. The core concept is only to record dynamic data if it will improve accuracy, either by omitting some files from dynamic data collection or by stopping data collection early. The time taken to record dynamic data is not always reported, but 5 minutes or a full execution of the sample are commonly reported (Table I). The cutoff for data samples may be based on sequence length rather than time, as in the case of Pascanu et al.’s work [8]. Data recording can be parallelised or run in an emulator, which gives slightly faster analysis (e.g. [5]), but reducing recording time per sample still enables more data to be collected in a shorter time.

Shibahara et al. [11] decide when to stop analysis for each sample based on changes in network communication, reducing the total time taken by 67% compared with a “conventional” method that analyses samples for 15 minutes each. Neugschwandtner et al. [21] used static data to determine dissimilarity to known malware variants using a clustering algorithm, and thus the implicit value of dynamic analysis. This approach demonstrated an improvement in classification accuracy by comparison with randomly selecting which files to dynamically analyse, or selecting based on sample diversity. Similarly, Bayer et al. [12] create behavioural profiles to try and identify polymorphic variants of known malware, reducing the number of files undergoing full dynamic analysis by 25%.

Each of these approaches, which calculate the expected benefit of dynamic analysis for each sample, may still execute an entire file. We look at cutting analysis short for all files and the resulting impact on accuracy with a view to a practical endpoint solution for detecting malware.

C. Recurrent neural networks for prediction

Time series data captures the changes in parameters as well as the parameter values themselves. For this reason we anticipate that a model capable of analysing time series data will give the highest classification accuracy. Recurrent neural networks and Hidden Markov models are both able to capture sequential changes, but RNNs hold the advantage in situations with a large possible universe of states and memory over an extended chain of events [22], and is therefore better suited to detecting malware using behavioural data.

RNNs use hidden layers which are deep in time, rather than space, to model changes between inputs in a sequence. Recurrent models have been employed successfully in the fields of natural language processing (NLP) and can be used to predict and generate text based on an initial short sequence of words or characters [13] or an image [23].

Weights in neural networks represent the importance of information travelling between two neurons. During training these weights are updated repeatedly to reduce the error rate in categorising the training data against its specified labels. Prior to the development of Long-Short-Term-Memory

| Reference | Reported time collecting dynamic data |
|-----------|--------------------------------------|
| [10]      | 30 seconds                            |
| [17]      | 3.33 minutes (200 seconds)           |
| [18]      | 5 minutes                             |
| [19]      | 5 minutes logging 10 times each sample|
| [16]      | Fixed time and 5-10 minutes mentioned but overall time cap not explicitly stated |
| [8]       | At least 15 steps - exact time unreported |
| [5]       | No time cap mentioned - implicit full execution |
| [20]      | No time cap mentioned - implicit full execution |
| [6]       | No time cap mentioned - implicit full execution |

TABLE I
REPORTED TIMES COLLECTING DYNAMIC BEHAVIOURAL DATA PER SAMPLE
(LSTM) cells by Hochreiter and Schmidhuber [24], RNN weights were liable to increase or decrease exponentially over long sequences [25]. LSTM cells can hold information back from the network until such a time as it is relevant or “forget” information, thus mitigating the problems surrounding weight updates. The success of LSTM has prompted a number of variants, though few of these have significantly improved on the classification abilities of the original model [26]. Gated Recurrent Units (GRUs) [27], however, have been shown to have comparable classification to LSTM cells, and in some instances can be faster to train [28].

Sequential data holds more information than data removed from its sequential context, and so provides additional information for difficult classification tasks such as identifying malware. Pascanu et al. [8], compared RNNs with Echo-State networks to “learn the language of malware” and detect reordered malware. The authors found that Echo-State networks outperformed RNNs for binary classification.

Kolsnaji et al [29] conducted extensive experiments with deep neural networks, including recurrent networks, to classify malware into families using API call sequences. By combining a convolutional neural network with LSTM cells, the authors were able to attain a recall of 89.4%, but do not address the binary classification problem of distinguishing malware from benignware.

Existing dynamic malware detection prioritises classification accuracy, even when trying to reduce the dynamic data recording time. Huang and Stokes [5] have demonstrated, using a very large dataset of 4.5 million training samples and 2 million testing samples, an accuracy of 99.6% can be achieved. In this paper, our goal is to examine whether it is possible to predict the likelihood that a file is malicious with a reasonable level of accuracy so that dynamic classification can be incorporated into an online malware detection system. For example, is it possible to have just 5% of samples misclassified within the first 3 seconds of execution? If not, with what degree of accuracy can we predict malware using 3 seconds of dynamic data? Does accuracy increase monotonically with more data or is there an optimal data sequence length for peak accuracy?

### III. System Description

Figure 1 outlines our proposed system for early malware prediction both in the training and classification phases. Our goal is to reduce the overall time taken by this pipeline, both for training and classification, whilst maintaining a high level of classification accuracy, to investigate the viability of using this system as part of an endpoint detection solution. We want to reduce the overall time for one sample to pass through the pipeline for classification. Per sample this equates to:

- Time in virtual machine (seconds or minutes) + preprocessing time (fraction of second) + classification time (fraction of second)

Predicting malware, if sufficiently quick, could enable dynamic malware detection prior to the file being executed in the desired environment (if deemed benign). In the future we could compare prediction accuracy over time with the time at which the malicious payload is executed. Few papers report the time for which data is recorded (see Table I), implying that data may be captured for the entire duration of file execution. Others imply fifteen [11] or five to ten minutes [16] before recording is stopped. Even if we take the five minute estimate, it is clear that the time in the virtual machine constitutes the bulk of the classification time per sample. Preprocessing, assuming it is automated, is likely to take fractions of seconds; to decorrelate or select features or convert data into some other format for example. Classification itself will be similarly quick. From [5] for example, we can deduce a classification time of 0.0056 seconds per sample using a FFNN. Therefore, reducing the time in the virtual machine by 50% will deliver a far greater reduction in overall time per sample than reducing either of preprocessing or classification time by half.

#### A. Data capture and dataset

The dataset comprises machine activity data collected in a virtual machine using the Cuckoo Sandbox malware analysis tool [50]. We captured 11 features each second for five minutes for 594 malicious and 594 trusted portable executable files provided by VirusTotal [31]. If the file finished execution before five minutes, data recording ended. The average sample recording time was 4.8 minutes, this took almost five days (95.4 hours) to collect for all 1188 samples. The dataset size
is consistent with dynamic malware detection research e.g. [6, 10, 32, 33, 16, 19, 34].

We capture numeric data, primarily machine activity data, which has advantages in preprocessing and model training time over categorical data. Many dynamic malware detection methods use system API calls e.g. [5, 35, 32, 8] which must be converted to numeric form before being fed into a machine learning classifier. Continuous numeric data does not need to be transformed. Often categorical data will be transformed into one-hot-vectors. One-hot-vectors are vectors of the size of the number of possible categorical values, in which each value is assigned an index; the value’s presence is represented by a ‘1’ in this vector and its absence by a ‘0’. Many possible values can cause the dimensions of the input data to be large, which makes the model slower to tune as the number of tunable parameters between the inputs and the first hidden layer of the network are proportional to the shape of the input data. Numeric data can represent many different values within a single input and so does not increase the network parameters as dramatically when capturing many kinds of data.

1) Features: Despite the rapid evolution of malware, the objective of malicious files if often shared with previously seen malware strains, for example gaining unauthorised access to a network or sending information out of a network. Benignware objectives are much broader, and may be defined as anything which is not malware. The alternative to behavioural analysis is to extract features from the malware code, which is not always available [9] and may be reordered [8] or obfuscated otherwise [16]. Similar to [36], we use machine activity data to monitor the file behaviour. The activities monitored represent all the machine activity data we were able to capture, and resulted in 11 data points each second for each sample.

The features captured were: total number of processes being executed, the maximum number of processes being carried out (derived from highest assigned process identification number), user and system CPU use (each as a percentage of total CPU capacity), memory use, swap use, number of packets received, number sent, number of bytes received, number of bytes sent and the number of milliseconds elapsed since the file began running. The milliseconds elapsed are included in the recurrent neural network even though the snapshots are taken at one-second intervals because of slight deviations that occur during data capture.

Figure 2 illustrates the relative frequency distributions of the values held by each of the 11 features between benign and malicious samples. These data represent values held across the entire data capture window (about five minutes per files). Table II gives summary statistics for the features. Bytes received and bytes sent exhibit the greatest contrast in distribution between malware and benignware, which may be indicative of sending data out of the network, a common goal of malware seeking to capture information. This is also consistent with Burnap et al.’s findings that network traffic was the most predictive feature for identifying malicious URLs [36]. Time-stamps exhibit near identical data, as expected, as data was collected at one-second intervals for both benign and malicious samples.

| Features                  | Benign     | Malicious  |
|---------------------------|------------|------------|
| Time                      | Min        | Max        |
| Total Processes           | 40         | 62         |
| Max. Process ID           | 2440       | 4092       |
| CPU User (%)              | 0          | 100        |
| CPU System (%)            | 0          | 63         |
| Memory Use (MB)           | 540        | 1545       |
| Swap Use (MB)             | 655        | 2026       |
| Packets Received          | 1,186      | 85,951     |
| Packets Sent              | 390        | 60,146     |
| Bytes Received (MB)       | 1.60       | 81.83      |
| Bytes Sent (MB)           | .05        | 101.87     |

Table II: Minimum, and Maximum Values of Each Input for Benign and Malicious Samples

B. Data Preprocessing

By using continuous numeric data, we avoid converting System API calls into large vectors as other dynamic behavioural analysis approaches are forced to (e.g. [5, 8]). We further implement two prepossessing measures aimed at speeding up classification and training time.

Prior to training and classification, we normalise the data to increase model convergence in training. By roughly keeping data between 1 and -1, the model is able to converge more quickly, as the neurons within the network operate within this...
C. Classifier training and prediction

We are interested in whether behavioural data about benign and malicious executables captured early on in file execution can give an indication as to whether or not the file is malicious, and the frequency with which this can be deemed accurate.

We anticipate that RNNs will yield the best predictive capabilities due to their ability to process sequential data, and compare it with other machine learning algorithms used for malware classification: Random Forest, Support Vector Machine (SVM), Naive Bayes, J48 Decision Tree, K-Nearest Neighbour and Multi-Layer Perceptron algorithms ([10], [6], [38]). Previous research indicates that Random Forest, Decision Tree or SVM are likely to perform the best of those considered.

As well as truncating the sequence length, we are also interested in the training and testing times of the models. Though this is secondary to recording time in the virtual machine, which is expected to provide the most scope for reducing processing time per sample. Support Vector Machines and RNNs, for example, take longer to train than random forests.

To simulate the manner in which an early detection tool might be used, we hold out 50% of the data for testing. In practice, the model used for predicting malware will be trained using some dataset but for the system to be of any use the model resulting from training must generalise well to detect new samples.

The accuracy achieved on the unseen test set classes best simulate the real-world situation of testing the classifier in detecting malware and provides be the benchmark for comparing algorithms. Malicious and benign samples were randomly assigned to either the training or test set to give two datasets comprising 594 samples (297 malicious and 297 benign). We use 10-fold cross validation on the training set to tune the hyperparameters of the RNN (see next Section III-D). Finally we test on the unseen test set.

The dataset is balanced, very minor imbalances may occur as a result of omitting short sequences during training and testing, so we use accuracy rather than F-Scores to measure model success. F-Scores help to evaluate the results of imbalanced datasets but are biased towards favouring true positives over true negatives.

We rely on Keras [39] with a TensorFlow [40] backend to implement the RNN and use the WEKA [41] machine learning toolkit to compare accuracy with other machine learning classifiers. To reduce training and testing time, the RNN is trained and tested on an Nvidia GTX 1080 GPU.

D. Recurrent neural network configuration

The choice of hyperparameters can be crucial in determining the success of a neural network. Further, the solution space(s) delivering good results can be quite small by comparison with the search space of possible configurations. Hyperparameters are often hand-chosen (e.g. [8], [42]) and some argue that the selection process resembles an art over a science [43].

In the domain of malware detection, it is important that this process be automated. Machine learning for malware detection has been a response to the need for automation, given the rate at which samples must be analysed. Malware evolves in response to new opportunities for gain and to evade detection. As such, the same RNN configuration may not consistently capture the relevant features for detecting malware. Automatic hyperparameter selection enables cheap and frequent re-searching of the possible configuration space in case for a better alternative. We conduct a random search of the hyperparameter space as it is trivial to implement, works well in a large search space and is more efficient than grid search at finding good configurations [44].

Some aspects of the model are chosen in advance for their potential to reduce model training and testing time. Weights are initialised at the initial uniform distribution outlined in [37] to speed up convergence. Between the input layer and output layer are hidden layers comprising GRU cells [27], chosen over LSTM cells for the potential reduction in training time [28]. The hyperparameters changed during random search and their possible values are indicated in Table III.

The number of hidden layers refers to the intermediate layers between the input and output layers. This is equivalent to model depth and can improve accuracy through layers of weights giving a hierarchical feature selection structure, but depth may also cause the data to overfit to the training data unless some regularisation technique is used. The number of GRU cells in the hidden layers lies between 1 and 500. Too few neurons may not capture enough information for accurate

range [37]. We achieve this by normalising around the zero mean and unit variance of the training data. For each feature, \( i \), we establish the mean, \( \mu_i \), and variance, \( \sigma_i \), of the training data. These values are stored, after which every feature, \( x_i \) is scaled:

\[
\frac{x_i - \mu_i}{\sigma_i}
\]

For the recurrent neural networks only, the data is divided into equal sized batches to increase the training speed over training one sample at a time. Random samples are omitted until the batch size is a factor of the dataset size.

To emulate the early stopping of data capture that we will use in the final system, we must truncate the data sequences. We select the desired sequence length, \( s \), which corresponds to the number of seconds of data captured in the virtual machine since we take a data snapshot each second. Sequences are then omitted or truncated relative to the chosen sequence length, \( s \). We remove all samples with sequences shorter than \( s \) and then truncate all data after time step \( s \) for the remaining sequences. For neural networks the sequences must all be the same length; we cannot pad out the short sequences using masking (as is possible with categorical data) because this is done using vectors of zero, which has a numerical significance so cannot be learnt as a class representing “no data” for continuous data. These sequences are omitted for the other classifiers to enable comparison between the algorithms.

\[x_i - \mu_i\]

\[\frac{x_i - \mu_i}{\sigma_i}\]
classification but too many can again cause the model to overfit to the training data.

Pascanu et al. [3] had some success using a bidirectional RNN, hypothesising that adversaries are likely to reorder malware and that bidirectional layers can help to reduce the efficacy of reordering by processing the time series data both progressing and regressing in time, but it may be that only progressive sequences are helpful for prediction.

The number of epochs gives the number of times the model weights are updated after the data is passed through the model, and the learning rate dictates how much the weights can be updated at each epoch. We allow the weight updating algorithm to be either stochastic gradient descent or the Adam optimiser [55]. Adam is a computationally efficient implementation of stochastic gradient descent which does not require tuning of its own hyperparameters, but stochastic gradient descent is commonly used in successful malware classification with neural networks (e.g. [3]).

We also vary regularisation techniques designed to prevent the model from overfitting to the training data. Dropout randomly removes a pre-specified proportion of units from the network [45]. Weight and bias regularisation can be used to penalise large weights (l1 regularisation) and to allow some weights to become large whilst penalising others (l2 regularisation).

The batch size varies between 1 and 59. The batch size denotes the number of samples that will be passed through the network in one propagation. Very small batch sizes will tend to increase the time that the network weights take to converge as the estimates of the best weight update are based on a very small, and therefore possibly unrepresentative, subset of the data. Larger batch sizes may equally contain redundant information if there is significant similarity between samples in the dataset [37]. 59 is the upper limit as this is the largest batch size possible for 10-fold cross validation with the training set of 594 samples, comprising 50% of the dataset.

Although there are only 10 parameters to tune, there are 4.5 trillion different possible configurations. As well as the hyperparameters above, we randomly select the sequence length of data. Although the goal is to find the best classifier for the shortest sequence, selecting an arbitrary length such as 5 or 10 seconds into file execution may not produce any models capable of high classification. We do not know whether a model will increase monotonically in accuracy with more data or peak at a particular time into the file execution. Randomising the sequence length used for training and classification reduces the chances of having a blinkered view of model capabilities.

IV. Experimental Results

A. Predicting malware using early-stage data

We want to determine whether machine learning can be used to detect malware during the early stages of file execution, whether there is a trade-off between time into file execution and detection accuracy. Each of the RNNs and other ML classifiers are fed an additional second of execution data to determine the predictive capabilities of the models.

The configurations of the best-performing RNNs are detailed in Table IV. These configurations were simply those which achieved the highest accuracy in 10-fold cross validation on the training set.

The trend of the accuracy metric is show in the Figure 3, the upper graph shows the accuracy trends on the training set, and the lower graph on the unseen test set. On the unseen test set, the RNNs significantly outperform the other classifier after 3 seconds of execution. Table V records the highest accuracy achieved on the test set by the RNNs and other algorithms.

The RNN cannot usefully learn from 1 second of data as there is no sequence so accuracy is equivalent to random guess. Using just 1 second of machine activity data, the random forest is able to classify 86% of unseen samples correctly. Using just 4 seconds of data the RNN correctly classifies 91% of unseen samples, and achieves 98% accuracy at 19 seconds into execution, whereas the next best algorithm reaches a maximum of 90% at any point during the first 30 seconds. The RNN improves in accuracy as the amount of sequential data increases.

The dataset size here is relatively small, but consistent with those used in similar research. To capitalise on the data available, we conduct a 10-fold cross-validation on the entire dataset, thus increasing the amount of data available for
Training (to 1050 for each fold) without reducing the test set to a single small batch of (possibly unrepresentative) samples. The 10-fold cross validation on the test set gives similar results (Table VI) to the test-train data, but demonstrates a more consistent trend to accuracy increasing with execution time.

Considering our initial question - is it possible to predict malware in 3 seconds? Results indicate that this may be possible with 90% accuracy. We do not achieve the desired 95% accuracy until 8 seconds. We can however use these figures as initial estimates for the trade-off between accuracy and the time into file execution. Beyond 15 seconds, the value of an extra second of data is much lower than during the initial 5 seconds.

### B. Impact of varying data snapshot intervals

In the following experiments we analyse the results of 10-fold cross validation across the entire dataset (1188 samples) as in Table VI for the reasons outlined in the previous section.

The results from Section IV-A indicate that the RNNs improve in classification accuracy with longer sequence lengths. This may be because later execution data gives a better indication of whether or not samples are malicious or because more data gives the model the ability to learn the nuances between malicious and benign machine activity better. We hypothesise that malicious activity begins as soon as possible once the executable begins running because this reduces the overall runtime of the file and thus the window of opportunity for being disrupted (by a detection system, analyst, or technical failure). More sequential data may enable the model to learn relationships between multiple features which is indicative of malware; activities A and B may be innocuous individually, but typically represent malicious activity when combined.

We vary the interval between feature sets to explore the question of whether later execution data or longer sequences have a higher correlation with model accuracy. Using data collected in the first minute of file execution, we increase the interval between data snapshots from 1-second to 2-, 3-, 4- and 5-seconds.
Figure 4 shows a general trend of increasing accuracy with more feature sets for each of the different time intervals, with the 3-, 4- and 5-second intervals outperforming the 1-second interval data. When mapped onto real time, as illustrated in Figure 5, it is clear that the one-second interval gains the highest accuracy the most quickly. With only four feature sets, data collected at 5-second intervals (95% accurate) outperforms the 1-second interval data (92% accurate). However in real-time, the 5-second data is using data 20 seconds into the file, at which point the 1-second data reaches an F-Score of 0.97.

Minutes and seconds are human constructs, and the choice of one-second intervals is just an arbitrary small amount of time chosen for these experiments. The pattern indicated here hints that the interval could be further reduced to improve accuracy. Further, it appears that sequence length, rather than a particular time into file execution gives the classifier the information required to make an accurate prediction. This trend of reducing sequence length to gain higher accuracy earlier is likely to have limitations, and should be investigated in future work.

### Table VII

| Average prediction | TN | FN | TP | FP | FN (%) | FP (%) |
|-------------------|----|----|----|----|--------|--------|
| 2                 | 0.10 | 0.73 | 0.89 | 0.32 | 15     | 12     |
| 3                 | 0.06 | 0.75 | 0.91 | 0.25 | 10     | 9      |
| 4                 | 0.04 | 0.80 | 0.95 | 0.23 | 10     | 8      |
| 5                 | 0.03 | 0.86 | 0.96 | 0.22 | 8      | 8      |

### C. Improving prediction accuracy with an ensemble classifier

Accuracy appears to increase with longer sequence lengths. We hypothesise that this is because longer sequences temper anomalous or unrepresentative feature sets. When adding 1 more second of data to a 2-second sequence, this data comprises one third of the data, but when increasing from 19- to 20-seconds, the new set is just 5% of the data classified. Analysis of false positives and false negatives shows that the model grows in confidence over time. Table VII shows how predictions move towards 0 and 1 on average for both correctly and incorrectly classified samples.

We attempt to temper the misguided confidence of the models in misclassifying executables as sequences grow longer. During training, neural network weights are adjusted according to some weight update rule (here Adam or SGD), but may not reach the globally optimal set of weights. Different hyperparameter configurations are likely to set different final weights for the network. For this reason, it may help to use an ensemble classifier. An ensemble classifier combines the predictions of multiple models. Ensembles can often yield better predictions than single models. Dietterich suggests that ensemble models can improve on individual models by averaging out poor predictions, mitigating the skew of models converged at local minima, or capture the complexity of a problem for which a single model will not suffice. An ensemble of models may improve classification since we have not fine-tuned the hyperparameters, but used a random search and are unlikely to have found the globally optimal hyperparameter set for classification. Configurations A, B, and C simply use the same model structure and are trained on different data for each additional second into the feature set, no single model in Table VI consistently achieves the highest accuracy across time steps, implying that each places different emphasis on different inputs and abstracted combinations of input data.

We create two ensemble classifiers, one to capture the differences between model configurations and one to capture the different learning emphases across sequence lengths. For the first, we adjust the batch size for models A, B and C to the batch size for model A (58), so that classification can be parallelised, and average the predictions for each sample. Adjusting the batch size gives slightly different accuracy scores to those reported in Table VI for models B and C.

The improvements, outlined in Table VIII are most marked...
Fig. 6. Frequency distribution of prediction values from sequence length 2 to 5 for all 10 folds. False predictions shown in green, True predictions shown in dark blue.

| Time (s) | Single Model (Config. A) | Ensemble of time sub-sequence models (Config. A) |
|----------|--------------------------|-----------------------------------------------|
| 2        | 87.08                    | 87.08                                         |
| 3        | 88.93                    | 89.01                                         |
| 4        | 91.08                    | 91.16*                                        |
| 5        | 91.67                    | 92.29*                                        |
| 6        | 93.01                    | 93.12*                                        |
| 7        | 94.43                    | 94.06*                                        |
| 8        | 94.68                    | 95.47*                                        |

TABLE VIII

Best accuracy of configurations A, B and C with ensemble classifier accuracy. * = increase in predictive confidence over best individual model is statistically significant at 5%; ** = at 1%

when classifying the shorter sequences. Though the gains in accuracy are modest, the increase in confidence for accurate predictions is statistically significant at 1% for sequences of 4 seconds or less and significant at the 5% level for sequences of 5 seconds. We calculate this score by taking the difference between the binary class, b, of the sample and the model prediction p. Accurate and confident predictions gain a score close to 1, and inaccurate or less confident predictions are closer to 0:

\[
\text{confidence} = 1 - |b - p|
\]

For the second ensemble, different models classify every sub-sequence of data longer than 2 feature sets, as well as the entire sequence itself before averaging the predictions. The number of models increases proportionally with sequence length. The number of classifications quadratically according to \(\frac{n(n-1)}{2}\) where \(n\) is the sequence length. For example, if the start and end of a sequence is represented as [start, end], and the model \(m\) used to classify a sequence of length \(n\) is represented by \(m_n\), then using 4 seconds of data we must perform the following classifications:

\[
m_4([1, 4])
\]

\[
m_3([1, 3]), m_3([2, 4])
\]

\[
m_2([1, 2]), m_3([2, 3]), m_2([2, 4])
\]

The results in Table IX indicate that again, the ensemble classifier, delivers marginal gains by comparison with the original model for almost every point into execution. The confidence of predictions is not as significant in the early stages as for the first (multi-configuration) ensemble. We believe that the short-sequence predictions are more divergent for the early models as a feature set is able to play a more significant role in a short sequence, thus having a stronger influence over the final prediction.

D. Interpreting the classifier

If the gains in accuracy for the ensemble classifier are due to differences in the features learned by the network, this could help to protect against adversarial manipulation of data. We attempt to interpret what configurations A, B, and C are using to distinguish malware and benignware. These
preliminary tests seek to gauge whether it is possible to analyse the decisions made by the trained neural networks.

Neural networks do not lend themselves to interpretation easily; Model A has nearly 23,000 trainable parameters and 112 nodes in the hidden layers. However, interpretation can help experts to interpret the results of algorithms and, as Ribiero et al. [48] argue, to trust those results.

We look at the impact of the 11 features at both the testing and training phases. The former is intended to give insight into what the model has learnt and the latter into the degree to which an RNN is able to find an alternative method of classification in the absence of a given feature.

Ribiero et al. [48] propose a system for black-box testing machine learning classifiers to interpret individual (or groups of) model predictions. This system is predicated on the ability to turn inputs “off”, which is easy in the case of categorical data for which the presence or absence can be represented as a binary value. It is more difficult to turn numeric data “off”, as zero bears a relationship to the set of possible numeric values for that input. Because we preprocess the data by normalising to the training set mean of zero and unit variance, the mean of the training samples for each feature is zero.

By setting the test data for a feature (or set of features) to zero, we can approximate the absence of that information between samples. We examine the impact on the overall accuracy, the true positive rate and the true negative rate of omitting (sets of) features.

We assess the overall impact of turning features “off” by observing the fall in accuracy and dividing it by the number of features turned off. A single feature incurring a 5 percentage point loss attains an impact factor of -5, but two features creating the same loss would be awarded -2.5 each. Finally, we take the average across impact scores to assess the importance of each feature when a given number of features are switched off.

Table X indicates a resemblance between models using configurations A and B, and a different emphasis for the model based on configuration C. A and B appear to use packets, bytes and System CPU use for differentiating the two classes, which reflects the difference in frequency distribution of the inputs shown by Figure 2. Model C appears to have a unlikely reliance on the precise time since the last data snapshot.

By altering the training data, we can see how robust configuration C is to training without the temporal difference data. The changes in accuracy detailed in Table XI see a maximum of 7 percentage points fall in accuracy and a gain of of 0.4 percentage points accuracy for the omission of Swap Use for Configuration B. By ranking the changes in accuracy we can see some clear patterns between the three configurations. Omission of swap use has a negligible impact on accuracy, whereas CPU system use unanimously incurs the greatest loss in accuracy. The total number of processes appears in the top 4 for A, B, and C though its individual impact score (when 1 feature is missing) does not generally appear in the top 4. The impact score increases relative to others as more features are omitted, this may indicate that total processes are combined with other inputs to create discriminating features, though the input is not highly impactful alone.

We can begin to understand the decisions made by the 4-second models using these methods, these preliminary findings indicate that we may be able to characterise model decisions. Characterising the decision process may in turn form the basis for analytics tools to help security analysts further understand malware behaviour and patterns within malware families.

V. LIMITATIONS AND FUTURE WORK

The experiments in this paper sought to explore the plausibility and limitations of online dynamic malware detection. Whilst the results presented indicate that the entire file is not necessary to achieve high classification, this is the first step in a series of questions leading to a possible end point solution to
Certainly it is necessary to choose a subset of all malwares of new malwares each day upon which to retrain the model. Training time and it may be necessary to select a sub-sample to be incorporated into the model. More data increases model formation that the model can glean from these samples needs to be tested with more data. If hundreds of thousands of new samples reported over time.

If the model is retrained daily this places greater emphasis on the hyperparameter selection being fast. Rather than global optimisation, it may be preferable to reach the highest accuracy possible in the shortest amount of time. Future work could examine methods for navigating the hyperparameter search space sequentially to reduce the time taken to find a good representation.

Finally, the robustness of this approach is limited if adversaries know that the first 5 seconds are being used to determine whether a file will run in the network. By planting long sleeps or benign behaviour at the start of a malicious file, adversaries could avoid detection in the virtual machine. We hypothesised that malicious executables begin attempting their objectives as soon as possible to mitigate the chances of being interrupted, but this would be likely to change if malware authors knew which parts of the file were being examined by security systems. One potential solution could be to train the model using such samples as we have just described, this is a technique suggested by [4] in the authors attempt to classify adversarially crafted static malware data. Another solution for a live classification system could monitor file behaviour over a sliding window (of 5 seconds for example) and would potentially be able to detect the malicious behaviour regardless of the time into the file at which it occurred, in which case the model should be retrained using machine activity capturing multiple processes at once together with API call sequences so that the calls from particular files can be distinguished from one another.

### VI. Conclusions

In this paper we have shown that it is possible to achieve an accuracy greater than 93% with just 4 seconds of dynamic data using an ensemble of recurrent neural networks and an accuracy of 98% in less than 20 seconds.

The best RNN network configurations discovered through random search each employed bidirectional hidden layers, indicating that making use of the sequence progressing as well as regressing in time aided distinction between malicious and benign behavioural data.

The RNN models outperformed other machine learning classifiers in analysing the unseen test set, though the other algorithms performed competitively on the training set. This indicates that the RNN was more robust against overfitting to the training set than the other algorithms.

To date this is the first analysis of the extent to which a file can be predicted to be malicious during its execution rather than using the complete log file post-execution.

### References

[1] C. Crime and I. P. Section, “How to protect your networks from ransomware.”

[2] VirusTotal, “Statistics - virustotal,” 2017. [Data for 26 April 2017].

[3] P. Vinod, R. Jaipur, V. Laxmi, and M. Gaur, “Survey on malware detection methods,” in Proceedings of the 3rd Hackers’ Workshop on computer and internet security (HITEHACK’09), pp. 74–79, 2009.
[48] M. T. Ribeiro, S. Singh, and C. Guestrin, “‘why should I trust you?’: Explaining the predictions of any classifier,” CoRR, vol. abs/1602.04938, 2016.

[49] M. E. Aminantoa and K. Kimb, “Deep learning in intrusion detection system: An overview,” 2016.