# Measuring the speed of the Red Queen's Race

**Richard Harang**[1] **and Felipe N. Ducau**[2]

[1] *Research Director, Sophos*

[2] *Principal Data Scientist, Sophos*

emails: richard.harang@sophos.com, felipe.ducau@sophos.com

**Abstract**

Security is a constant cat-and-mouse game between those trying to keep abreast of and detect novel malware, and the malware authors attempting to evade detection. The introduction of the statistical methods of machine learning into this arms race allows us to examine an interesting question: how fast is malware being updated in response to the pressure exerted by security practitioners? The ability of machine learning models to detect malware is now well known; we introduce a novel technique that uses trained models to measure "concept drift" in malware samples over time as old campaigns are retired, new campaigns are introduced, and existing campaigns are modified. Through the use of both simple distance-based metrics and model confidence measures, we look at the evolution of the threat landscape over time, and show that – from the perspective of a machine learning model – the rate of innovation in the malware landscape is low: the majority of concept drift occurs due to the loss of familiar samples, rather than the development of novel samples. In addition, from the perspective of a machine learning model, individual families are remarkably stable, with very little change from month to month in their feature representation. In combination, these two finding appear to suggest that techniques for evading machine learning based models do not appear to be widespread.

*Key words: Malware evolution, Concept drift, Fitted Learning, Deep learning*

## 1 Introduction

Malicious content developers are constantly creating new pieces of malware and introducing changes in their software. Some of these changes are related to variations in the functionality or the introduction of new capabilities. Others are part of a continuous effort to avoid detection by the cybersecurity industry. To achieve this goal, there are several techniques

BlackHat USA 2018, August 4-9, 2018, Las Vegas, NV, USA

that malware developers use, from simple ones such as addition of superfluous content to the files or packing, to advanced polymorphism techniques.

On the other side, anti-malware solutions attempt to identify these new or modified malware programs with high precision in the least amount of time possible following their introduction. Traditionally, malware detection has been done by looking for specific attributes of the programs, such as code, binary sequences, file properties, system calls, and others, which are unique to a malware sample or campaign. This approach is commonly referred as "signature-based detection". Usually, these patterns (signatures) to be matched against files are hand-written by security analysts as soon as they can analyze some novel piece of malware that is not being detected. These signatures are carefully designed to have high specificity and a very low false positive rate.

Because of the reactive nature of the signature-based approach, in recent years, statistical techniques driven by machine learning research have been introduced for the task of malware detection in an attempt to improve detection coverage for the window between the introduction of new malware samples and the writing of specialized signatures to detect them. These machine learning approaches look for "features" – numerical statistics derived from the files – or various combinations of those features which are indicative of malicious activity in a data-driven way. Machine learning (ML) based malware classifiers are trained with several millions malware and benign samples, and try to extract patterns in the data in a way that generalizes to unseen samples outside their training set.

Even though these ML classifiers have proven to be powerful at spotting malware that they have never seen while training, the changing nature of the threat landscape causes their performance to degrade with time, as old campaigns are retired or updated and new ones are developed. This change in the data distribution of a machine learning model is called *concept drift*. In this work, we will study the way in which deep learning based malware classifiers react to changes in the underlying malware distribution and explore ways to exploit this information to help us reason about the change in the threat landscape over time. In particular, we will present a method to estimate the rate at which malware campaigns are generated and retired, and a method to identify clusters of potentially novel malware families. For this, we will use a special kind of neural network architecture introduced in [7] that allows us not only make predictions about the maliciousness of a given file, but also obtain an estimate for the confidence of those predictions. We also examine the manner in which changes in malware impact their feature representation as a tool to investigate the degree to which various campaigns and families change over time from the perspective of our model.

The remaining of this paper is organized as follows: Section 2 is an overview of existing techniques and approaches to detect malware with deep learning models. We will also make a brief introduction to the idea of *concept drift* along with a survey of proposed methods to detect it in the case of malware detection as well as suggested mitigation measures; Section

3 introduces the concept of Fitted Learning, an architectural modification to traditional neural network models developed by Kardan and Stanley [7] that allows us to measure the confidence of the predictions made by them; In section 4 we will explore how can we use the change in the confidence of our ML model predictions as an analytic tool to study the rate of change in the malware distribution over time. With this tool we will look at the change in the malware landscape on real data collected over the period January 2017 to April 2018 in aggregate. In 4.1 we will focus our analysis in two popular campaigns available in that data, *WannaCry* and *HWorld*; Section 5 explores how the distance of a given sample to a fixed reference set can be used as a proxy to understand how different malware families change over time. Finally we will analyze the relationship between the confidence of a prediction for a given sample and the distance of this sample to the training and present a visual method to identify possible novel malware campaigns.

# 2 Related Work

## 2.1 Malware detection with neural networks

In recent years, several approaches have been proposed to tackle the problem of detecting malicious software using neural network techniques with promising results. In this work we will focus particularly on analysis of Windows Portable Executable (PE) files based on static features; information that can be extracted from the binary without having to run it.

Even though most of the work in the area uses similar techniques, the main difference between them relies in the approach used to represent the input files. It is not the goal of this section to make an exhaustive summary of the work in the field, but mention the ones that relate closer to ours. Cakir and Dogdu [1] make use of a disassembler to retrieve the opcodes of the executable files and then a shallow network based on WORD2VEC [8] to embed them into a continuous vector space. Afterwards, they train a gradient search algorithm based on Gradient Boosting Machines for the malware classification task. The drawback of this approach is that the necessary unpacking of the programs to be able to extract the opcodes is not always a straightforward task. Raff et al. [10], were able to learn a malware classifier for PE binaries by feeding the raw sequence of the first 300 bytes from the PE header of the files to feed-forward and recurrent architectures. Raff et al. [9] extended this approach by developing an architecture that combines feed-forward, convolutional and recurrent neural networks, to ingest the entire raw sequence of bytes of the files, which can be several million bytes long. The work of Saxe and Berlin [11] is the closest to ours, both in terms of the architecture used as well as the feature representation of the files. The authors used a feed-forward neural network trained to predict if a given PE file is malicious or benign by looking at aggregated features extracted directly from the binary files as well as some other meta-data obtained via static analysis. This deterministic aggregation, applied in a pre-processing step, causes the final representation of a file to be of fixed size

independently of the original size of the binary. Even though there is loss of information in the feature extraction process, it has shown to work well in practice. Furthermore, this compact representation of the binaries allows for fast training of the networks.

### 2.2 Concept drift in malware distribution

In the field of statistical learning, *concept drift* refers to the change in the relationship between input features and the target variable for the underlying problem over time. This phenomena is also known as *covariate shift*, *dataset shift* or *nonstationarity*. This change in the underlying statistical properties of the task is problematic for machine learning classifiers, since one of the main assumptions under which they are trained, is that the data that is going to see in deployment comes from the same data generation process as the one it was trained on. In other words, these algorithms assume that the data is sampled from a stationary distribution. Intuitively, the malware data generation process does not fit this assumption well. On the contrary, we know that malware development is dynamic, and changes over time by nature to avoid detection.

The most basic approach to deal with a changing data distribution is to retrain the model as frequently as possible. Kantchelian et al. [6] propose practical considerations when dealing with a malware landscape which is in constant change, particularly they propose to train an ensemble of classifiers in which each of them is only tracking one family and then combine their predictions to produce a single classification. They also propose a framework for human analysts to work along ML models to overcome the drift in adversarial scenarios. The work of Deo et al. [3] propose the use of *Ven-Abers* predictors for assessing the quality of the classifications to help determine a more efficient retraining strategy than naively retraining the model continuously. It does so by having a well calibrated model that helps them determine when the quality of the predictions is decaying.

The *Trascend* framework, developed by Jordaney et al. [5] was proposed to identify aging of classification models during deployment before the model's performance starts to decay. The proposed technique to assess the quality of the machine learning model is the conformal evaluator (CE) framework which is inspired on Conformal Predictors (CP) [12]. CE uses $p$-value calculations together with the output labels provided by the algorithm under evaluation in order to detect concept drift.

Our work differs from the mentioned ones in the sense that we are not aiming to overcome the problem of concept drift, but instead, use that information as a proxy to understand and reason about the change in the malware ecosystem over time.

### 2.3 Malware evolution

Also relevant to our study, the work by Calleja et al. [2] analyzes the evolution of malware between the 1980s to 2016 from a software engineering perspective. From the source code of

151 malware samples they computed measures of code size and quality as well as estimations for their development costs. This analysis is complementary to ours, since their study focuses on a fixed set of malware samples from different points in time while we look at the amount of malware and number of campaigns introduced over time.

## 3 Fitted learning

Vanilla implementations of deep learning classifiers do not have a built-in capability to determine the confidence of their predictions. If we train a model to distinguish between pictures of dogs and cats, and then we give it a picture of a whale it will still issue a prediction (dog or cat), even though the picture (sample) we are showing to the model is far away from the pictures (samples) that it has seen during training. This is due to the fact that conventional neural networks have a tendency to *overgeneralize* outside the range of their "knowledge".

Figure 1 exemplifies this phenomena in a visual way. Suppose we train a classifier to distinguish between the red and blue samples that live in a 2 dimensional space. By learning the decision boundary shown in the background and predicting everything to the top and right as being blue and everything to the bottom and left as being red it achieves an accuracy of 100%. The issue appears when we try to predict samples which come from a third and different class that the classifier has never seen - black samples in the right-hand plot. If these samples are introduced after training time, our naive classifier will predict that all of them are blue with high probability just because they are on the blue side of the learnt decision boundary.

In order to avoid this kind of behavior, the fitted learning framework [7] learns in a way that captures the data distribution of the training data and prevents the network from overgeneralizing outside the train distribution of samples. The main idea behind this technique is called *competitive over-complete output layer* (COOL) and it consists of assigning more than one output to each of the possible classes of a classifier while, at the same time, forcing those outputs to compete with each other. Forcing this competition in the output layer of the network produces a partitioning of the input space where the outputs only agree in the regions of the input space that are close to the training instances. What this means in practice is that now our classifier is learning to differentiate between classes, while capturing the distribution of the training instances of each class at the same time.

The authors of [7] define a *neuron aggregate* as a collection of output units in the network, called *member units*, which all attempt to learn the same concept. The practical way of implementing the competition between member units is to use a softmax layer on top of all the member units of all the neuron aggregates. Because of the nature of the softmax function - if one output is larger, all the remaining ones are necessarily smaller - all the member units of the same neuron aggregate are *competing* against each other. The number

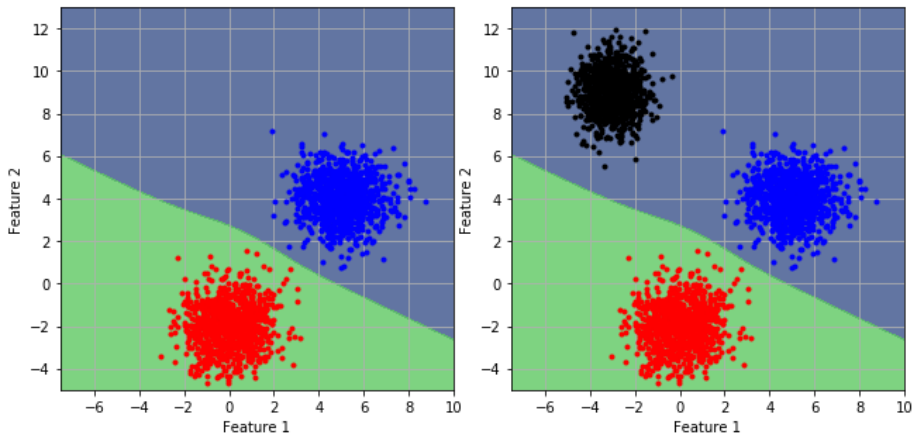


Figure 1: Example of the over-generalization phenomena. *Left:* the classifier is trained to distinguish between blue and red dots and learns the decision boundary shown in the background. *Right:* if a new type of samples appear at test time (black dots), they will be classified as being blue, even though they are far away from the distribution of blue samples.

of member units per class $w$ is the *degree of over-completenes (DOO)* of the network.

During training of the network, all the member units of the same neuron aggregate are trained with the same value - 0 if the label is not from the same class as the neuron aggregate and $1/w$ otherwise. At inference time all the outputs of the member units of the same class are combined via multiplication in order to get the final output. If we want to produce a probability estimate for each of the classes, each aggregate output can be multiplied by $w^w$. It is worth noting that, with fitted learning, it is no longer the case that the sum of the outputs, even after re-normalization, add up to one. In this work, we will define the *confidence* $c$ of a model for a given sample, as the sum of all the outputs of the network. This magnitude can be interpreted as the total amount of probability mass that the model is assigning to *any class* for the sample. It is easy to see that the maximum possible value of $c$ is obtained when all the outputs for a given neuron aggregate are $1/w$, resulting in an estimated probability for the given class of one. Both samples that are close to the decision boundary and those which are far from the training distribution will end up having low confidence scores.

In Figure 3 we show the result of training the same network, except for the output layer, on the same problem as before with fitted learning with a DOO of 10. The plot of the left shows in green the regions in which the classifier's confidence is high ($c \geq 0.85$) and the right depicts the regions for which the classifier assigns more than 85% of probability of a sample being of a given class. Note that in the previous case (Figure 1) the classifier assigned probabilities higher than 95% for the entire 2-dimensional space.

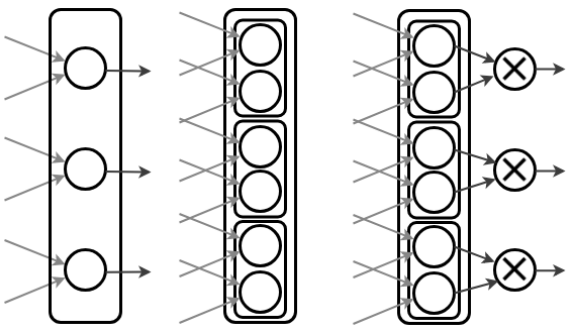

Figure 2: Output layer of a conventional neural network versus COOL. *Left:* A conventional output layer with three neurons to learn three concepts. *Middle:* An over-complete output layer for the same task during training with DOO of 2. Each inside block is a neuron aggregate with 2 member units where both are trying to learn to identify the same class. *Right:* The same COOL architecture at test phase, the output of all the units within a neuron aggregate are combined through multiplication to obtain the final predictions.

# 4 Using confidence to examine changes in malware distribution

As previous research has shown, machine learning models can learn to predict whether a given file is malicious or benign, with high accuracy and reasonable false positive rates. This is true as long as the underlying data distribution does not change. When deployed in practice, these models start off performing as well as expected, but over time the accuracy starts to decay and the false positive rate begins to increase. Intuitively this is mostly caused by the introduction of new malware campaigns, attempts of malware developers to maintain their software unnoticed, and retiring of old campaigns that were easy for the cybersecurity industry to detect.

Even though trained models performance diminishes over time, it does it in a gradual way. In this section we will look at how the confidence of a simple feed-forward network trained to distinguish between malicious and benign PE files changes over time. The details on the network architecture, hyperparameters and training procedure are described in the Appendix A.[1]

For this experiment we collected 3 million PE binaries per month, from January 2017 to April 2018 observed on a threat intelligence aggregator, resulting in 16 disjoint datasets $\mathcal{D} = \{D_{Jan17}, D_{Feb17}, \ldots, D_{Apr18}\}$. The collection mechanism ensures that if a sample is present in the dataset corresponding to the month $m$, $D_m$, then the first time that that

[1]The code used for the experiments will be uploaded to `https://github.com/inv-ds-research/red_queens_race`

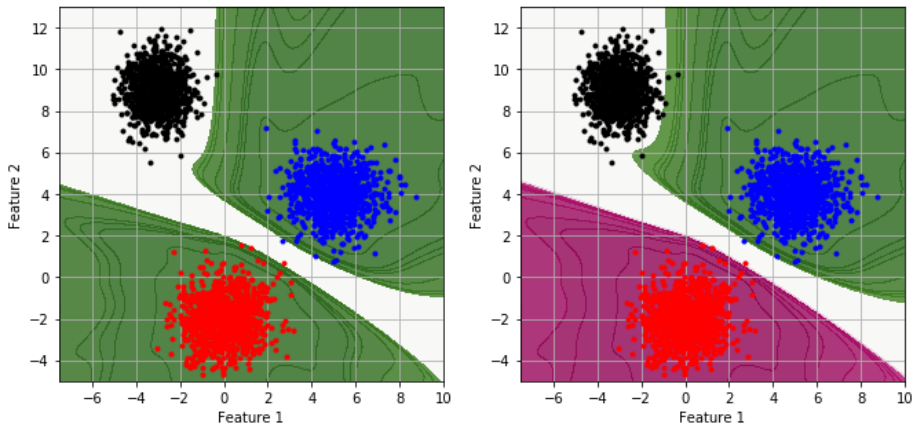


Figure 3: Confidence and decision boundaries for a network trained with COOL output layer. *Left:* regions of the input space for which the confidence assigned by the classifier is larger than 0.85. *Right:* Decision boundaries for $p(y|x) > 0.85$ for each of the classes.

file was observed in the aggregator happened during that month. The data for each month contains both benign and malicious PE files in a proportion close to 70/30 respectively.

We proceeded to train one model per month of data. The model architecture is a 5 layers feed-forward network, with COOL and DOO of 16. The same architecture and hyperparameters were fixed for every training run, resulting in a set of 16 trained models with entirely different data $\mathcal{M} = \{M_{Jan17}, M_{Feb17}, \ldots M_{Apr18}\}$. Each of the models is only aware about kind of malware that was present in the dataset corresponding to that month. If a malware campaign was first observed in month $m$, all the models $M_{i<m}$ have never seen a sample of that campaign.

Next, for a model trained on data from month $m$, $M_m$, we run the samples in each of the data sets of the subsequent months and compute the confidence that the model assign to each of them, repeating the process for each model. This is, with the model trained with January 2017 data we score all the future samples from that model's perspective (from February 2017 onward). February 2017 model scores samples from March 2017 to April 2018, etc.

Instead of looking at the confidence scores directly, we define three confidence buckets of interest: low confidence $c \leq 0.1$; medium confidence, $0.1 < c < 0.9$ and high confidence $c \geq 0.9$. The low confidence bucket contains samples that are substantially different from the bulk of training data because they are new to the model, or variants with large statistical differences from the ones the model was trained on, or samples that are close to the decision boundary. On the other end, high confidence samples are very similar - from the model perspective - to existing samples in its training set. They could be near-duplicate files,

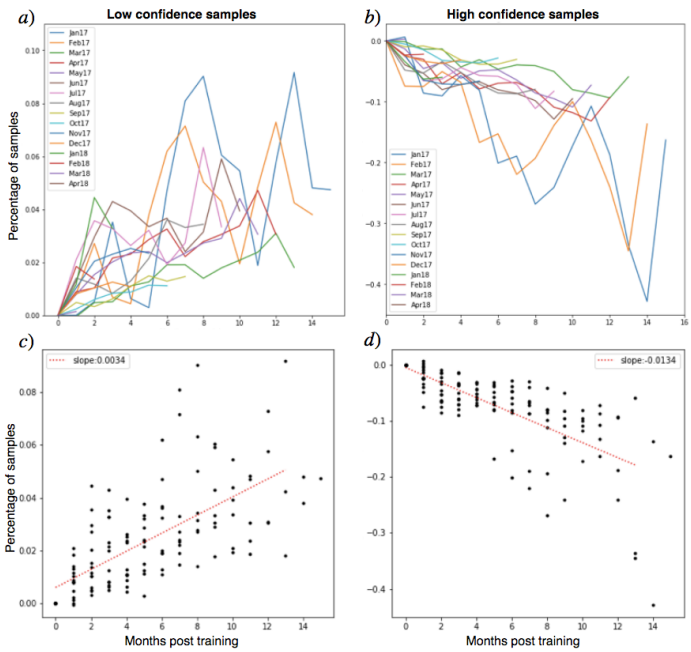


Figure 4: Percentage of samples in the *a)* low and *b)* high confidence buckets over time. Linear regression fits, *c)* for the percentage of samples in the low confidence bucket and *d)* for the sample proportion in the high confidence zone

hashbusters, file infectors applied to different files, files with large amounts of "borrowed" code, etc.

Figures 4a) and 4b) show the percentage of samples in the low and high confidence buckets respectively for each of the models when evaluated on data from the future. Even though the curves are noisy, there is a clear increasing trend for the percentage of samples in the low confidence zone and a decreasing tendency for the number of samples that fall in the high confidence bucket.

In Figures 4c) and 4d) the same data is shown as a scatter-plot and the best linear fit is computed for each case. The linear regression coefficients are 0.0034 and -0.0134 for the percentage of dataset samples in the low confidence and high confidence buckets over time respectively. The low confidence percentage rises at a rate close to 1% per quarter, which can be interpreted as the amount of novel malware that is introduced in the population over time. On the other hand, the percentage of high confidence samples falls at around 4% per quarter, this trend is guided by families that were well known to the model but being retired from the overall population.

| | Jan17 | Feb17 | Mar17 | Apr17 | May17 | Jun17 | Jul17 | Aug17 | |
|---|---|---|---|---|---|---|---|---|---|
| Wannacry | 0 | 0 | 1 | 12 | 542 | 1156 | 1461 | 1365 | |
| HWorld | 0 | 207 | 10622 | 6294 | 2773 | 1861 | 10175 | 12583 | |
| | Sep17 | Oct17 | Nov17 | Dec17 | Jan18 | Feb18 | Mar18 | Apr18 | Total |
| Wannacry | 1267 | 908 | 2735 | 2153 | 1517 | 3090 | 2444 | 1721 | 20372 |
| HWorld | 5659 | 10273 | 7500 | 8285 | 6944 | 15031 | 23352 | 16870 | 138429 |

Table 1: Number of samples for WannaCry and HWorld malware in our 3 million datasets over time.

### 4.1 Case study: WannaCry and HWorld families

In this section we will focus our analysis in two malware families from 2017: *WannaCry* and *HWorld*[2]. Particularly we will look at the confidence that each of our models trained with monthly data assign to *all* the samples from these two families. In the subset of data used for this experiment there are a total of 20,372 samples related with the WannaCry campaign and 138,429 HWorld samples. The monthly counts for each of the families is shown in Table 1 and the method to determine if a given sample belongs to these families is defined Appendix B.

As in the previous section, low confidence samples are those samples that are far from the training distribution of the model or close to the decision boundary between benign and malicious classes. In figure 5 the results of this experiment are presented. It is important to note that January and February '17 models have not seen any instances of WannaCry during training. The same is true for January '17 model with respect to the HWorld family. Nevertheless, we observe in the plots that 55.9% of the WannaCry samples are found to be high confidence when scored with January model. For those samples the detection rate is higher than 99.9% which suggests that (1) the machine learning approach generalizes well to unseen samples, (2) that the WannaCry campaign has features in common with previously known families, most likely previous *Ransomware* campaigns, which are well-attested to in the training data, and (3) our model confidence scores are empirically supported, given the high detection rate on the high confidence subset. For the HWorld family the case is different, only 2.75% of the samples from this family are scored by the January '17 model as being high-confidence, which implies that the characteristics of the binary files from this campaign are substantially different than the features present in the January 2017 data. However if, as above, we classify only those high-confidence samples from future months, we obtain a true positive rate of 94.3%.

The higher percentage of low confidence WannaCry samples compared with the percentage of low confidence samples in the HWorld campaign suggest a higher turnover in variants of former family. Finally, comparing these plots with the counts in table 1 we see that the

[2]The *HWorld* family predates 2017, however we observed no samples of it in our January data.

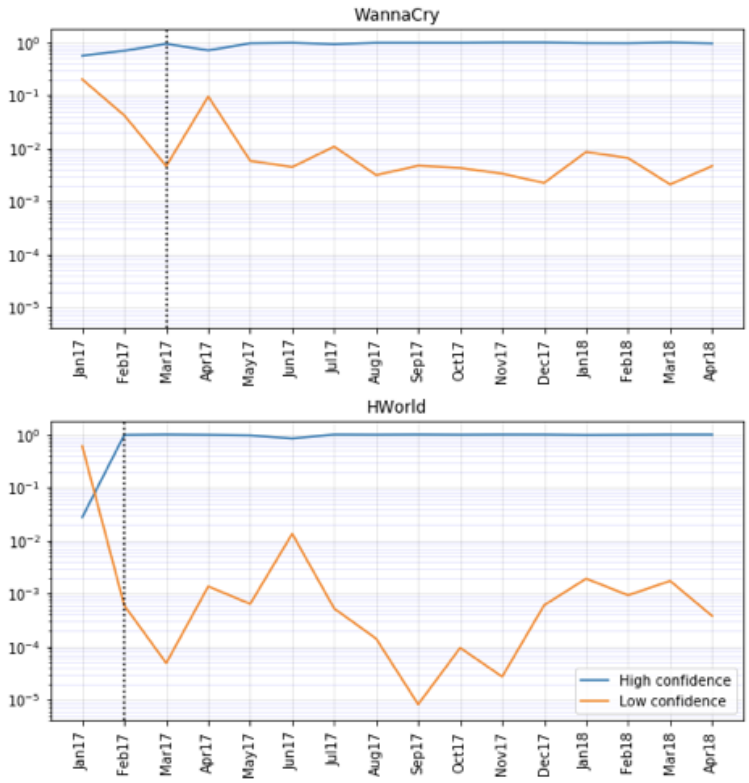


Figure 5: Percentage of high and low confidence samples for WannaCry (top figure) and HWorld (bottom figure) families when scored with models trained in different months of data. Dotted vertical lines indicate the first time a sample for the given family was present in our training sets.

amount of samples that our model needs to achieve a high percentage of high-confidence samples is relatively small, the model trained in September 2017 saw during training 6.7% (1,365) of WannaCry samples and 9% (12,583) of HWorld samples, but was able to score as high confidence 97.65% and 99.96% of the total samples respectively.

## 5 Distance and confidence as measure for novelty

In this section we present early work on using distance measures to track development of individual families over time as well as to potentially detect novel families.

As described in the introduction, the first step of presenting files (including portable executable files) to an ML model is converting them to a feature representation: a list of

numbers describing various properties of the file. Features can be as simple as simply the length of the file, require detailed parsing, such as counting the size of individual sections, or be format agnostic, such as calculating the entropy of a file. These feature representations can be viewed as points in high-dimensional space, locating a specific file, and distances can be computed between them, using such metrics as the Euclidean distance $d(x_i, x_k) = ||x_i - x_k||_2$.

By tracking the distance between members of the same family over time, we can obtain a measure of how the families change over time from the perspective of an ML model. For a given family in a given month, we find the nearest neighbor (in feature space) of the same family that exists in the January 2017 data set. If a family is exhibiting systematic change over time, such as the introduction of a range of significantly novel variants with large differences in the statistics that make up their feature representation, we would expect their distances to show a marked and consistent increase over time. In figure 6 we examine the results for the nine most popular non-generic malware families that were present in all months of our data. With the potential exception of the Ramnit-A worm, no sample exhibits any significant drift over time. If the $99^{th}$ percentile is examined, most families show at most minor fluctuations, peaking around July 2017, however these nearly immediately revert to baseline in the following month. When viewed through the lens of the feature representation, malware families show remarkably consistent feature space appearance over time.

In Section 3 we observed that there are two cases that can lead a sample to be scored by a model as having low confidence when using the fitted learning framework: (1) when the sample is far away from the training data distribution and (2) when the sample is close to the decision boundary between classes in feature space. In order to disentangle this effect we can look at both the confidence and the distance of the sample to the training set. As above, we will define this distance $d_i$ as the Euclidean distance from sample $i$ to its nearest neighbor in the training set, $d_i = \min_{k, x_i \neq x_k}(||x_i - x_k||_2)$.

Figure 7 shows the confidence assigned by January '17 model in the x-axis and distance to January '17 training set in the y-axis for the data in January and July '17. The contour lines in the plot indicate the density of the January '17 data in these regions, with each line indicating a line of equal probability through the data (estimated via a Kernel Density Estimator). Even though the defined distance does not have a straightforward interpretation, we can use it to compare across plots. As we would expect, most of the samples from the month in which the model was trained (left plot) lie on the high confidence zone and the distances tend to be small.

When plotting July data the case is different because of concept drift: the overall confidence of the model is small and the distances to the training data grow. If we focus on the samples in the medium and low confidence buckets with higher distances we would expect to find novel samples. Manual inspection of high density clusters in this region indeed

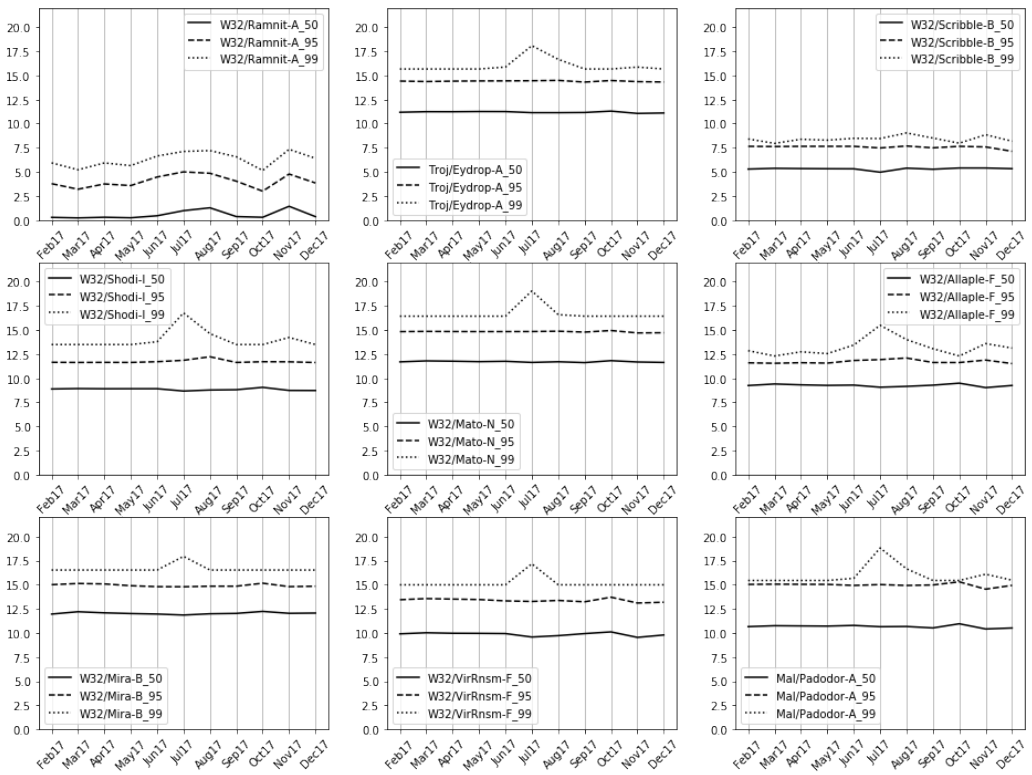


Figure 6: Family distances from the nearest neighbor of the same family in the January 2017 data set. Lines indicate the median ($50^{th}$ percentile), $95^{th}$, and $99^{th}$ percentile of distances for a given family in a given month to the nearest member of the same family in the January 2017 data.

contain samples from malware families that were not present in the training data, as well as new clusters of existing 'generic' detections: *Mal/Behav-238* (1,468 samples), *Mal/VB-ZS* (7,236 samples), *Troj/Inject-CMP* (6,426 samples), *Mal/Generic-S* (318 samples) and *ICLoader PUA* (124 samples) [3] along with several clusters of benign samples.

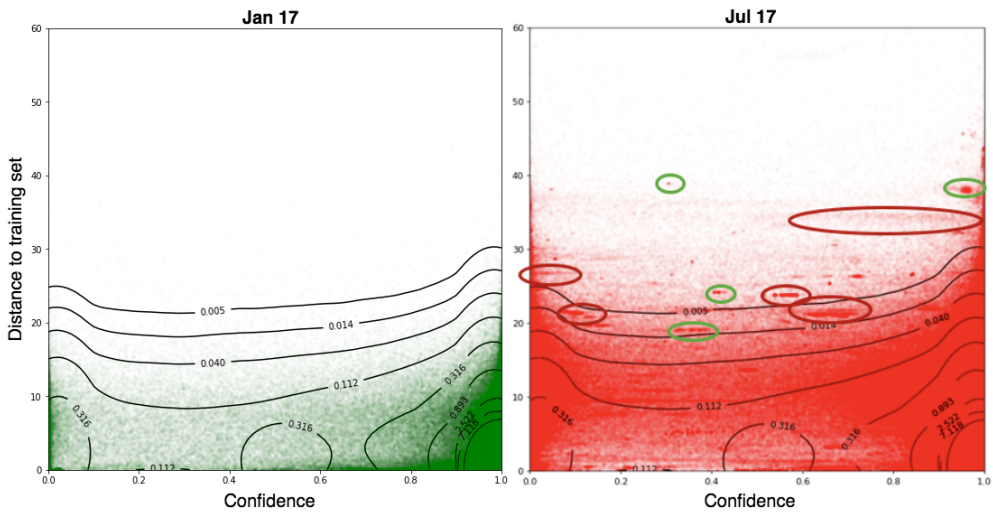


Figure 7: Confidence vs. distance to training set plots for *Left* January 2017 (data in which the model was trained) and *Right* July 2017. High density clusters for malware (in red) and benign (green) samples highlighted.

## 6 Conclusions and future work

Machine learning and signature-based methods of detecting malware are complementary in its nature. On the one hand, signature-based detection produces high confidence results with very low false positive rates. On the other, neural network models can detect new types of malware for which no signature could have been written yet.

Integrating the concept of confidence from the fitted learning framework to neural network models allows us to analyze population and family drift, and measure the rate at which new malware is introduced and retired from the threat landscape. Furthermore, we showed that it is possible to correctly classify novel malware samples with high accuracy

[3] Family names obtained from Sophos AV detection names.

even with an "old" model if we issue predictions for those which the model assigns a high confidence score.

Finally, we introduced a technique to visualize the threat landscape that allows us to find clusters of interest for finding potentially novel malware campaigns that can be used to guide manual inspection of files.

We expect to extend this work in the future by studying the relationship between the confidence of the samples and the change in accuracy had we had the samples labeled. We would expect to find that the lowest confidence samples are most informative ones to the model to reshape its decision boundaries and therefore the ones that we would get the highest return in accuracy if we labeled them. Another possibly fruitful research direction would be to use clustering techniques to try to identify automatically novel malware campaigns on data by confidence and distance to training set information. Finally, we believe it is important to explore computationally more efficient methods to disentangle the causes of scoring low confidence observations in the fitted learning framework.

## Acknowledgements

Special thanks to Richard Cohen for sharing his expertise on malware genealogy, Joshua Saxe for his fruitful discussions and the rest of the Data Science team at Sophos.

## References

[1] Cakir, Bugra ; Dogdu, Erdogan: Malware Classification Using Deep Learning Methods. In: *Proceedings of the ACMSE 2018 Conference*. New York, NY, USA : ACM, 2018 (ACMSE '18), S. 10:1–10:5. – URL `http://doi.acm.org/10.1145/3190645.3190692`. – ISBN 978-1-4503-5696-1

[2] Calleja, Alejandro ; Tapiador, Juan E. ; Caballero, Juan: A Look into 30 Years of Malware Development from a Software Metrics Perspective. In: *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, URL `https://doi.org/10.1007/978-3-319-45719-2\_15`, 2016, S. 325–345

[3] Deo, Amit ; Dash, Santanu K. ; Suarez-Tangil, Guillermo ; Vovk, Volodya ; Cavallaro, Lorenzo: Prescience: Probabilistic Guidance on the Retraining Conundrum for Malware Detection. In: *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. New York, NY, USA : ACM, 2016 (AISec '16), S. 71–82. – URL `http://doi.acm.org/10.1145/2996758.2996769`. – ISBN 978-1-4503-4573-6

[4] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[5] Jordaney, Roberto ; Sharad, Kumar ; Dash, Santanu K. ; Wang, Zhi ; Papini, Davide ; Nouretdinov, Ilia ; Cavallaro, Lorenzo: Transcend: Detecting Concept Drift in Malware Classification Models. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC : USENIX Association, 2017, S. 625–642. – ISBN 978-1-931971-40-9

[6] Kantchelian, Alex ; Afroz, Sadia ; Huang, Ling ; Islam, Aylin C. ; Miller, Brad ; Tschantz, Michael C. ; Greenstadt, Rachel ; Joseph, Anthony D. ; Tygar, J. D.: Approaches to Adversarial Drift. In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. New York, NY, USA : ACM, 2013 (AISec '13), S. 99–110. – URL http://doi.acm.org/10.1145/2517312.2517320. – ISBN 978-1-4503-2488-5

[7] Kardan, N. ; Stanley, K. O.: Fitted Learning: Models with Awareness of their Limits. In: *ArXiv e-prints* (2016), September

[8] Mikolov, T. ; Sutskever, I. ; Chen, K. ; Corrado, G. ; Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. In: *ArXiv e-prints* (2013), Oktober

[9] Raff, E. ; Barker, J. ; Sylvester, J. ; Brandon, R. ; Catanzaro, B. ; Nicholas, C.: Malware Detection by Eating a Whole EXE. In: *ArXiv e-prints* (2017), Oktober

[10] Raff, E. ; Sylvester, J. ; Nicholas, C.: Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In: *ArXiv e-prints* (2017), September

[11] Saxe, J. ; Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. In: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, Oct 2015, S. 11–20

[12] Vovk, Vladimir ; Gammerman, Alex ; Shafer, Glenn: *Algorithmic Learning in a Random World*. Berlin, Heidelberg : Springer-Verlag, 2005. – ISBN 0387001522

[13] Žliobaitė, Indrė ; Pechenizkiy, Mykola ; Gama, João: An overview of concept drift applications, 2015

# Appendix A Experimental details

## A.1 Fitted learning example

For the example presented in section 3 all the samples come from 2-dimensional Gaussian distributions as follows:

$$\text{Red} \sim \mathcal{N}(\mu = [0, -2], \sigma = [1, 1]) \tag{1}$$
$$\text{Blue} \sim \mathcal{N}(\mu = [5, 4], \sigma = [1, 1]) \tag{2}$$
$$\text{Black} \sim \mathcal{N}(\mu = [-3, 9], \sigma = [0.75, 1]) \tag{3}$$

Both the base and fitted learning models are 2 layer feed-forward neural networks with a hidden dimension of 200 units and tanh nonlinearities.

## A.2 Experimental models

All the models used in this work are feed-forward networks of depth 5 with layer sizes [768, 512, 512, 256], ELU nonlinearities and batch normalization. The output layer and loss function are the same as described in [7] with DOO=10. The input features of size 1024 are the same as the ones described in [11]. Networks were trained using Adam optimizer with a learning rate of $10^{-3}$ for 50 epochs.

# Appendix B Wannacry and HWorld family identifiers

For determining if a given malware sample belongs to one of the families in the analysis we use the following detection names from Sophos AV scans:

- **Wannacry**
  - Troj/Ransom-EMG
  - Mal/Wanna-A
  - Troj/Wanna-C
  - Troj/Wanna-D
  - HPMal/Wanna-A
  - Troj/Wanna-E
  - Troj/Wanna-G
  - Troj/Dloadr-EDC
  - Troj/Agent-AWDS

  - Troj/Wanna-H
  - Troj/Wanna-I
  - Troj/Ransom-EMJ
  - Troj/Wanna-J
  - Troj/Wanna-
- **HWorld**
  - W32/HWorld-A