Proceedings of the 50th Hawaii International Conference on System Sciences | 2017

Discovering Malware with Time Series Shapelets

Om P. Patri
University of Southern California
Los Angeles, CA 90089
patri@usc.edu

Michael T. Wojnowicz
Cylance Inc.
Irvine, CA 92612
mwojnowicz@cylance.com

Matt Wolff
Cylance Inc.
Irvine, CA 92612
mwolff@cylance.com

Abstract

Malicious software ('malware') detection systems are usually signature-based and cannot stop attacks by malicious files they have never encountered. To stop these attacks, we need statistical learning approaches to identify root patterns behind execution of malware. We propose a machine learning approach for detection of malware from portable executable (PE) files. We create an 'entropy time series' representation of the content of each file, and then apply a unique time series classification method (called 'shapelets') for identifying malware. The shapelet-based approach picks up local discriminative features from the entropy signals. Our approach is file format agnostic, can deal with varying lengths in input instances, and provides fast classification. We evaluate our method on an industrial dataset containing thousands of executable files, and comparison with state-of-the-art methods illustrates the performance of our approach. This work is the first to use time series shapelets for malware detection and information security applications.

1. Introduction

The evolving volume, variety and intensity of vulnerabilities due to malicious software ('*malware*') call for smarter malware detection techniques [32]. Most existing antivirus solutions rely on signature-based detection, which requires past exposure to the malware being detected. Such systems fail at detection of new malware which was previously unseen ('*zero-day attacks*') [7]. A new zero-day was discovered each week on average in 2015 [2].

Effective statistical learning approaches can automatically find root patterns behind execution of a malicious file to build a model that can accurately and quickly classify new malware [3,4,29-32,35]. We propose a new approach for malware detection from Microsoft portable executable (PE) files [26] using an advanced time series classification approach, which can pick up *local discriminative features* from data. Existing approaches to classification that use entropy analysis [16] or wavelet energy spectrum analysis [36] often only use global properties of the input signal.

Time series shapelets. Our approach is based on *time series shapelets* [39]. A *shapelet* is a subsequence from the input (training) time series, which can discriminate between classes in the data. Intuitively, shapelet-based approaches focus on finding local discriminative features within the data, and once these are found, the rest of the data is discarded. Shapelets have been used for a wide range of time series data mining tasks [11,15,21,22,23,27].

Entropy representation. Modern malware may contain sophisticated malicious code hidden within compressed or encrypted files [5,16]. For instance, parasitic infections and injected shellcode[a] often rely on packed (compressed) code segments with concealed malicious code. Entropy analysis [5,34,37] has been used to detect such attacks. As observed by Lyda et al. [16], sections of code with packing or encryption tend to have higher entropy. We choose to represent each PE file by an '*entropy time series.*' We get byte-level content from each file, make non-overlapping chunks of the file content, and compute entropy of each chunk. This gives us the entropy time series (ETS) representation[b] – one time series for each file. Given labeled training data (both benign and malware files), our aim is to classify a test file as safe or malicious. Thus, we frame the malware detection task as a time series classification task.

Challenges. There are multiple challenges involved with malware detection from complex executable files found in the wild [4,13,29,31]. The input data varies in structure, nature, patterns and lengths of time series. Our data has ETS lengths ranging from a couple of data points to more than a hundred thousand data points. Some subsections of ETS show multiple rapid changes in entropy while others are largely flat or zero.

---

[a] Modeling polymorphic shellcode is an intractable problem [33]

[b] 'Time series' is not meant in the literal sense of time, but in the statistical sense of sequential data that is not i.i.d. (independent and identically distributed)

URI: http://hdl.handle.net/10125/41898
ISBN: 978-0-9981331-0-2

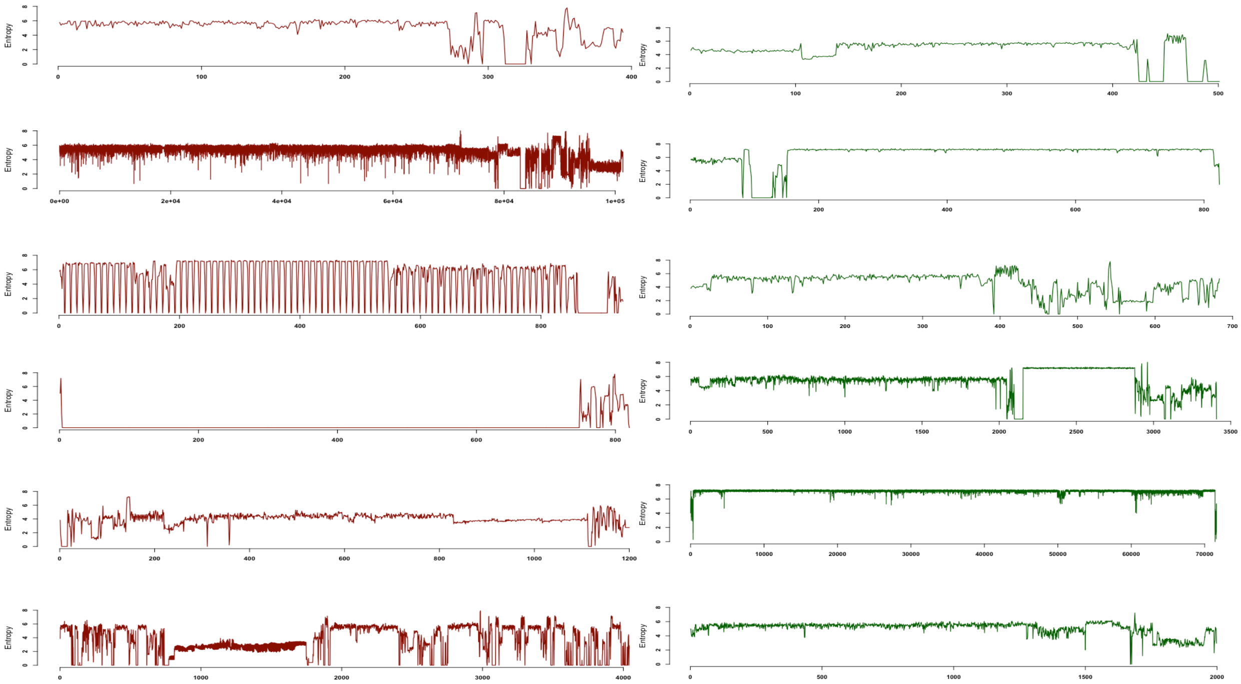

Figure 1. Entropy time series from malicious/malware files (left/red) and benign/safe files (right/green). Note the wide variation in lengths and patterns among the data.

There is no obvious hint or giveaway within the structure of the data differentiating the benign files from the malicious ones. To illustrate the variety within our data, we plot a few randomly selected ETS instances from each category (malicious and benign PE files) in Figure 1.

Contributions. In this work, we introduce and motivate the use of shapelets in the cybersecurity domain for malware detection and file content analysis. We convert the malware detection problem into a time series classification task, and demonstrate the efficacy of our shapelet-based classification method on entropy time series representations of executable files. From our results, we observe that shapelets can learn useful *discriminative* patterns from a very small fraction of the training data and perform accurate classification. Further, we show that the distance from the shapelet can be used as a single (or additional) feature for any supervised learning task using an existing classifier.

The rest of the paper is organized as follows. Section 2 presents related work on malware detection and Section 3 motivates the application of shapelet-based approaches to the computer security domain. Section 4 presents relevant background on how the shapelet extraction algorithm works. Section 5 describes our malware detection approach. Section 6 delineates the dataset and evaluation methods, and discusses the observed results. We summarize our work and conclude in Section 7.

2. Related work

Malware detection. Malicious software has become rampant in our daily lives, affecting our computers and mobile devices. Static analysis methods examine the content of a file without executing it. As reported in Moser et al. [17], deception techniques such as code transformations, obfuscations and metamorphism can overcome static approaches used by many virus/malware scanners.

Siddiqui et al. [32] provide an overview of malware detection approaches using file features. A common source of file features is N-grams (of mnemonics of assembly instructions) [35]. N-grams are sequences of bytes of a certain length, and contain bytes adjacent to each other. These approaches typically process an extremely large number of N-grams. Wavelet transformations [37] are another source for file features. Bilar [6] proposed using mnemonics of assembly instructions from file content as a predictor for malware. Statistical machine learning and data science methods [9] have been increasingly used for malware detection, including approaches based on support vector machines, logistic regression, Naïve Bayes, neural networks, deep learning, wavelet transforms, decision trees and k-nearest neighbors [4,8,13,16,29-32,37].

Entropy analysis of malware. Entropy analysis [5,16,34,36,37] is an effective technique for aiding detection of malware by pointing to the possible presence of deception techniques. Despite polymorphism or obfuscation [19], files with high entropy are more likely to have encrypted sections in them. When a PE file switches between content regimes (i.e. native code, encrypted section, compressed section, text, padding), there are corresponding shifts in its entropy time series [37].

Usually in entropy analysis of files for malware detection, either the mean entropy of the entire file, or entropy of chunks of code sections in the file (as in our approach) are computed. This simplistic entropy statistics approach may not be sufficient to detect expertly hidden malware, which for instance, may have additional padding (zero entropy chunks) to pass through high entropy filters. In our approach, we develop a machine learning approach to automatically identify patterns of changes in entropy, which are indicative of malicious file behavior.

3. Motivation

Novel contribution to malware detection. Even though several machine learning classifiers [30] and digital forensics methods [10] have been used for file analysis and malware detection, this work is the first to use time series shapelets in the computer security domain in general, and for malware detection in particular. We believe shapelets are inherently well-suited to malware detection as they identify local discriminative patterns, and identifying these patterns helps identify unknown malware. Shapelets are also appropriate for classifying new time series very quickly as shown in our experiments, as (i) they perform classification on test data very quickly, and (ii) they don't need a lot of training data to learn the shapelet patterns.

Illustrated example. As an example consider the following shapelet (in Figure 2) – the red portion showing the extracted shapelet subsequence within its time series. This was a discriminative shapelet extracted from our dataset, details of which are in Section 6.2. This shapelet belongs to a time series instance from the malware class, and might be indicative of malicious behavior with the power to predict similar (or dissimilar) subsequences in new time series. At a broad view, this shapelet signals that a sharp drop in entropy could possibly be indicative of malicious code behavior. In Figure 2, the file had constant, high entropy for a large portion and then there was a large entropy drop – both of which could be suggestive of a code obfuscation section. This feature would have been detected if the drop occurred at a different location in the file, but might not have been detected if the scale of the drop was different (shapelet feature matching is invariant to horizontal translation but not to vertical dilation). These invariance properties are well-suited for signaling the potential presence of obfuscated code to a malware detector: the existence, rather than the location, of the obfuscation is what's important (hence, horizontal invariance), and moreover the drop from high entropy to different levels of low entropy could reflect shifts to different types of content (plain text, native code, padding, etc.; hence, vertical non-invariance).

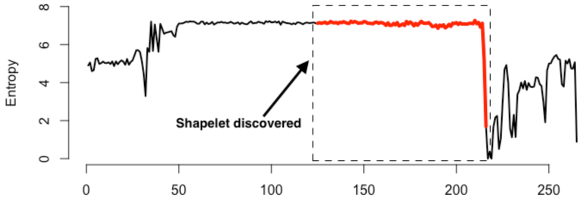

Figure 2. A shapelet found from our dataset

Why shapelets? Despite the existence of many other approaches for time series classification [38], we use shapelets in our work, as (i) they find *local* and *discriminative* features from the data, (ii) they impose no assumptions on the nature of the data unlike autoregressive or ARIMA time series models [38,39] and they work even on non-stationary time series, (iii) they work on data instances of different lengths (unlike popular classifiers such as support vector machines, feed-forward neural networks, and random forests in their standard forms), (iv) they are easy to interpret and visualize for domain experts, and (v) they have been shown to be more accurate than other methods for some datasets [11,12,15,18,20-24,27,39]. Shapelet-based time series classification methods have been used in diverse application areas including shape mining of historical objects [39], bioinformatics and medical data analysis [11,27], physical activity recognition [27], automobiles and manufacturing [20,24], oilfields [21,22], smart-grids [23], and robotics [18].

4. Background on shapelets

In this section, we explain how the time series shapelet extraction algorithm works. We consider a binary (two-class) classification scenario. Time series shapelets were first proposed by Ye and Keogh [39] and there have been optimizations on the initial method to make it faster or more advanced [12,15,18,20,24,27].

First, we introduce the basic brute-force shapelet extraction algorithm for time series classification [39]. As shown in Algorithm 1, the input dataset, D, contains a set of time series instances (TS) and their corresponding labels (Label). Given all the input time series and labels, the algorithm generates *candidate* subsequences of all possible lengths from all locations across all the training time series. The upper and lower length bounds (maxL and minL), as well as the 'length step size', can be specified as parameters.

As we know, the Euclidean distance between two vectors S and $\overline{S}$ (each of length l) is defined by:

$$dist(S,\overline{S}) = \sqrt{\frac{1}{l}\sum_{i=1}^{l}(S_i-\overline{S}_i)^2}$$

and the distance between a candidate subsequence S of length l and a time series T is calculated as:

$$dist(S,T) = min_i\ dist(S,T_i^l)$$

where T_i^l is a subsequence at position i and length l within the time series T. Thus, it is the minimum Euclidean distance obtained by sliding the candidate subsequence against the (longer) time series instances, and recording the smallest distance observed at any position. The distance from each candidate to all the data instances in the training set are calculated.

A good shapelet candidate should be able to *split* the training instances into two sets (predicted as malware, and predicted as benign) based on distance from itself. Each shapelet candidate would make this split differently, and we denote the two predicted sets created by a shapelet through a split (sp) variable. Given a candidate and split pair, we use an information gain (InfGain) metric to determine the best of the candidates, as:

InfGain = Entropy(Before) – Entropy(After),

where Entropy is the total entropy of the dataset, thus, the sum of entropy of instances in the two classes. Entropy(Before) is the entropy of the original dataset when split into two classes (based on the label of each instance), and Entropy(After) is the entropy of the split generated by the shapelet, thus, the sum of the entropy of the two predicted sets in the split. A shapelet is the candidate with the maximum information gain across the split, which means it is the most discriminative subsequence found from training data. A distance threshold metric is also learnt when identifying the shapelet.

For complex datasets, one shapelet may not be enough to get a clear separation between the two classes. In these cases, multiple shapelets are extracted and arranged in the form of a tree, similar to a decision tree classifier. The tree has shapelets in the internal nodes and predicted class labels in the leaf nodes. As a test instance traverses this tree, a decision on whether to follow the left or right child at each (internal) node can be made by comparing the distance threshold of the shapelet to the distance of the test instance to the shapelet. The test instance is classified when a leaf node is reached.

We use the Fast Shapelets [27] approach. This is a state of the art supervised shapelet mining approach, which uses random projections to get approximate, nearly optimal solutions. Fast Shapelets works with a change in data representation – instead of working on the raw time series, it works on a dimensionally reduced symbolic representation Symbolic Aggregate Approximation (SAX), more details of which can be found in [14]. This process reduces dimensionality of original data and improves generalization of the classifier (helping to avoid overfitting). The average time complexity for shapelet extraction (training) in the Fast Shapelets algorithm is $O(mn^2)$ where m is the number of time series and n the length of time series.

```
Given D = [TS, Label]
max_gain ← 0
for len = minL to maxL
    Candidates ← GetCandidates(TS, len)
    for each cand in Candidates
        create split sp from cand
        gain ← InfGain(D, sp)
        if gain > max_gain
            max_gain ← gain
            shapelet ← cand
        end if
    end for
end for
```

Algorithm 1. Basic shapelet discovery algorithm

5. Approach

In this section, we describe our proposed approach for malware detection in detail.

5.1. Proposed classification approach

An overview of our classification approach is shown in Figure 3, and it can be summarized in three steps (after we have access to the training and testing files and class labels).

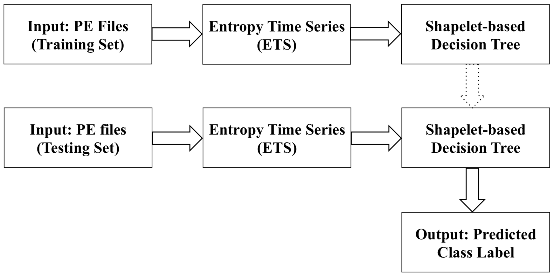
Input: PE Files (Training Set)
Entropy Time Series (ETS)
Shapelet-based Decision Tree
Input: PE files (Testing Set)
Entropy Time Series (ETS)
Shapelet-based Decision Tree
Output: Predicted Class Label

Figure 3. Overview of our classification approach

Step (1) is **entropy time series creation**. In information theory, entropy (more specifically, Shannon entropy [28]) is the expected value of information contained in a message. Generally, the entropy of a variable refers to the amount of disorder (uncertainty) in that variable (for example, the entropy is zero when one outcome is certain).

We have multiple labeled instances of both malicious and benign files. We convert each of them to the entropy time series representation. We perform the same preprocessing on the test data files as well (before classification). To do this, we consider contiguous, non-overlapping chunks of the file content and compute entropy for each chunk. The size of each chunk is set at 256 bytes (but can be changed). The decimal representations in these chunks are numbers from 0-255 (hexadecimal equivalent of 00-FF) and correspond to assembly instructions. Entropy is usually defined as:

$$H = -\sum p(x) \log p(x)$$

Here, the probability p(x) of each byte (ranging from 0 to 255) is the count (or frequency) of how many times it occurs within the current chunk (normalized by the chunk size). So, the entropy of each chunk is defined as:

$$H = -\sum_{i=0}^{255} f_i \log_2 f_i$$

Entropy only takes into account the probability (here, frequency) of observing a specific outcome, so the meaning of the bytes themselves does not matter here (only their frequencies). The entropy of each chunk is measured in bits and is a non-negative number between 0 and 8, both inclusive.

Step (2) **shapelet extraction** (Training). Once we have the input data in time series format, we can perform shapelet extraction. We use the Fast Shapelets [27] method for this step. This process gives us the shapelets themselves, and a decision tree based on shapelets, along with their corresponding distance thresholds. Each internal node of the tree is a shapelet, while the leaf nodes contain final class label predictions. In the shapelet extraction process, we can choose the minimum and maximum lengths of shapelets we desire to be picked. Alternatively, if the minimum length is set to 2 and the maximum length to the length of the smallest time series in the dataset, then the method can be completely parameter-free.

We can see that our approach gradually reduces the files to entropy time series and then to shapelets, in the process of extracting the most relevant and discriminative features out of the data.

Step (3) **shapelet-based classification** (Testing). In the final step of our approach, we use the shapelet-based decision tree to predict the class label of a new test file, presented in the ETS format. The distance of the new time series to the shapelet in the root of the tree is calculated, and based on whether the distance is less or more than the threshold provided, we follow either the left or right subtree. This process goes on iteratively until we have found a path to a leaf node, and then a class prediction for this instance is made. Similarly, we classify all the test data instances.

5.2. The 'distance from shapelet' feature

Apart from treating shapelets as a stand-alone classifier, we can also view it as a method to obtain a single feature (or multiple features) to assist an existing classifier that uses many features. This feature is based on the distance of a time series instance to a shapelet discovered from the training data [15,39]. To reiterate, the distance of a time series (typically longer) to a shapelet subsequence (typically shorter) is obtained by sliding the shapelet across the time series and noting the minimum Euclidean distance between the shapelet subsequence and the corresponding time series subsequences.

Intuitively, this is similar to compressing the whole length of a time series instance to just one scalar value – *its distance from a shapelet*. We may have multiple shapelets extracted from a dataset, and so, the distance from each shapelet can become a feature by itself, or they can be aggregated to form a single feature. From our evaluation, we notice that this distance is indeed a powerful feature, and even though it reduces the dimensionality of the data greatly, it still carries a lot of predictive power.

5.3. Enhancements using the PE file format

Even though our proposed approach is file format agnostic, our intention is to classify PE files, and thus, our approach can benefit from additional information about the PE file structure, outlined by Pietrek in [26]. In our approach, we use the complete file content to form the ETS. Using selective information from the PE file format for specific tasks could make our approach more effective.

For instance, PE files could be analyzed on a section-by-section basis. Inside the `Image_Section_Header` structure are two fields helpful for identifying padding (non-functional code): `SizeOfRawData` (size of the section as it exists on disk) and `VirtualSize` (how much memory should be reserved for the section when it's loaded from disk to memory). `SizeOfRawData` is almost always a multiple of the disk sector size, which is traditionally 512 bytes, whereas `VirtualSize` tends to be (but isn't strictly required to be) the exact number of bytes occupied by the section. For example, a section that contains the bytes "ABCDEFG" will have a SizeOfRawData of 512 bytes and a VirtualSize of 7 bytes, and it will be stored in a file as the bytes "ABCDEFG" followed by 505 bytes of padding (usually zeroes but sometimes a simple repeating pattern). Thus, one might discard the 505 bytes of padding and expect the overall performance of our algorithm to improve.

In another example, regions of the file expected to have a specific type of content (for instance, relocations or *'fix-ups'*, compressed content such as PNG images, and cryptographic data such as digital signatures) could be discarded or handled separately. Of course, an adversary could use the discarding technique to his or her advantage (by lying about the VirtualSize), but more generally, information about the PE file structure could certainly be used to enhance the performance of our algorithm, and is an exciting direction for future research.

6. Evaluation

We describe our dataset, evaluation method and results in this section. Due to page limits, we had to skip some of our evaluation results and plots in this paper (such as experiments on class ratio balanced data) – but they can be found in the supporting webpage [1] for the paper.

6.1. Data

We use a real industrial dataset of nearly 40,000 Windows portable executable files. Half of them are labeled as malware, and the other half as benign. These files contain a wide range of executables in their type and nature. We focus our experimental evaluation on a subset of 1,599 files from this set, which has also been used by Wojnowicz et al. [36,37]. We realize our enterprise dataset is proprietary, and we suggest researchers wishing to reproduce our work to use similar file content data from the Kaggle page[c] of the Microsoft Malware Classification Challenge at WWW 2015/BIG 2015.

6.2. Experimental methodology

Shapelet parameters. We perform several experiments on many different ratios of random training-testing splits from the data, ranging from using 10% to 90% of the data for training. Any data instance that is not used for training is held out to be used in the test set. Unless otherwise specified, we set the minimum shapelet length to be found as 5, and the length step size for shapelet search as 1. As the Fast Shapelets approach we use is a randomized algorithm, the accuracy results vary slightly when the method is run multiple times. Unless otherwise noted in our evaluation, we report the mean accuracy of 10 runs (along with the standard deviation) of the shapelets classifier on a dataset. We do not normalize the time series in our dataset – all the entropy values lie between 0 and 8 (both inclusive) and a pilot test showed no significant difference in performance between the z-normalized and unnormalized datasets.

File size groups. A challenge dealing with our dataset is that the lengths of the ETS (directly leading from the lengths of the files generating them) are vastly different – ranging from just a few data points to more than 100,000, with a mean length of 3,725 and a median length of 920. For convenience, the authors in [37] proposed grouping the dataset into file size groups based on ETS lengths. We follow the same convention and group all ETS of lengths between $[2^j, 2^{j+1})$ into a group J=j. We focus our experiments on a set of 1,599 files satisfying J=5 (ETS lengths between 32 and 64), thus limiting our shapelet search to lengths between 5 and 32 (the smallest time series in the dataset restricts the maximum length of shapelet that can be found). This J=5 set has also been picked for evaluation using wavelet transformations by Wojnowicz et al. [37], and thus we picked it to allow comparisons with their wavelet-based method. They proposed a Suspiciously Structured Entropic Change Score (SSECS) feature for malware classification, and we compare the efficacy of our proposed Distance from Shapelet (DFS) feature to that of SSECS.

An extreme experiment. A key observation here is that our shapelet learns from data very quickly – with only a small percentage used for training. We attempted an extreme experiment to illustrate this. We randomly sampled 1% of our original (near 40,000 time series) dataset with the restriction that the ETS lengths were between 100 and 10,000 (which still includes about 80% of our full dataset). We extracted shapelets (minL=5, maxL=100) and built a shapelet-based decision tree from this 1% set (containing 321 time series from 179 benign and 142 malware files). Nine shapelets were extracted in this case to form the decision tree. We used the rest (99%) of the data (31,980 time series) for testing, and obtained a test accuracy of 67.3%, which is comparable to the accuracy obtained by the wavelet transformation and entropy approach in [37] using 80% of the data for training. The first and most discriminative shapelet found from this experiment, which might be indicative of malware, is shown in Figure 2.

---

[c] https://www.kaggle.com/c/malware-classification

Feature representations. To use existing popular classifiers and evaluate the utility of the DFS feature, we create a feature representation of the dataset (with DFS, SSECS and statistical features). A set of eight basic statistical features (also used in [37]) captures information about each ETS. They are:

- Mean
- Standard Deviation (std. dev.)
- Signal-to-noise ratio (mean/std. dev.)
- Maximum entropy
- Fraction of signal with high entropy (>=6.5 bits)
- Fraction of signal with zero entropy
- Length of time series
- Square of length of time series

6.3. Shapelet-based malware classifier

We present results of our classification approach on our chosen dataset of 1,599 files in Figure 4. We compare our shapelet-based method to a baseline classifier, which assigns every test data instance to the label of the majority class in the training dataset. Our dataset is quite imbalanced across the two classes (as shown by the baseline classifier), but we also experimented on a resampled dataset with balanced class ratio, with results shown in Figure 6.

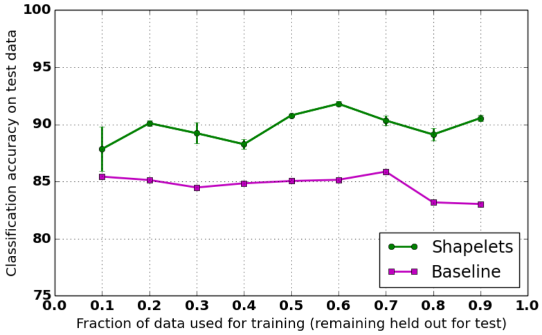

Figure 4. Classification (test) accuracy

The mean accuracy varies from 87.84% (using 10% of data for training) to 90.55% (using 90% of data for training). We can observe that even on using just 10% of the training data, our classifier has quickly gained a lot of predictive power. We experiment on 9 train-test ratios of the dataset, the fraction used for training ranging from 0.1 to 0.9 (in increments of 0.1). Every time we have a new train-test split ratio, we randomly choose whether an instance in the original dataset goes to the training or testing test. Due to this random splitting, the randomized nature of shapelets itself and possibly overfitting, it may sometimes happen that even though the fraction of data used for training increases there is no improvement (or a dip) in the prediction accuracy on the test set. The difference between our algorithm's performance and the baseline is much stronger for the balanced dataset as seen in Figure 5.

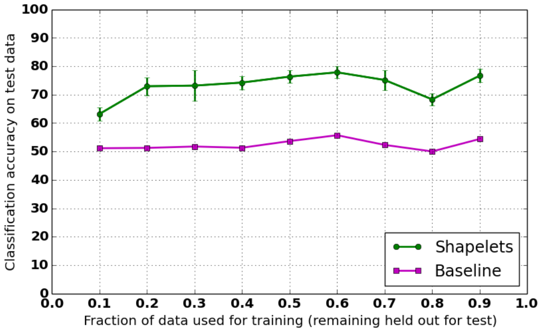

Figure 5. Classification accuracy on balanced dataset

6.4. Distance from shapelet feature

We intend to explore the utility of the *Distance from Shapelet* (DFS) feature, possibly as an additional feature in an existing classifier. This existing classifier can be any classifier, and we use these three – support vector machines (SVM), logistic regression classifier (LR) and random forests (RF). To implement these classifiers, we used the scikit-learn Python package [25] with their default parameters. It is likely possible to optimize the parameters of each classifier and get higher accuracy, but our main goal is a relative comparison of classification accuracy across our chosen feature sets. We compare performance of each classifier on these feature sets:

1- **SSECS** – using only the SSECS score
2- **DFS** – using only the DFS feature
3- **SSECS+Stat** – using SSECS score and eight statistical features defined earlier
4- **SSECS+DFS+Stat** – using SSECS score, DFS, and eight statistical features.

The evaluation results are shown in Figure 6 and Table 1 (results on balanced data available on the supporting webpage [1]). For the DFS feature above in our evaluation, we only use the distance from the first (most discriminative) shapelet extracted – this node is the root node of the shapelet decision tree and all instances must pass through the root. For all three classifiers (RF, LR and SVM), SSECS is always outperformed by DFS. When statistical features are added to SSECS, the accuracy increases. However, adding the DFS feature still improves the overall accuracy in a large number of cases. Thus, the DFS feature is useful (as a single feature) to add to an existing classifier. For SVMs, DFS alone outperforms all three other combinations in cases where the fraction of data used for training is small.

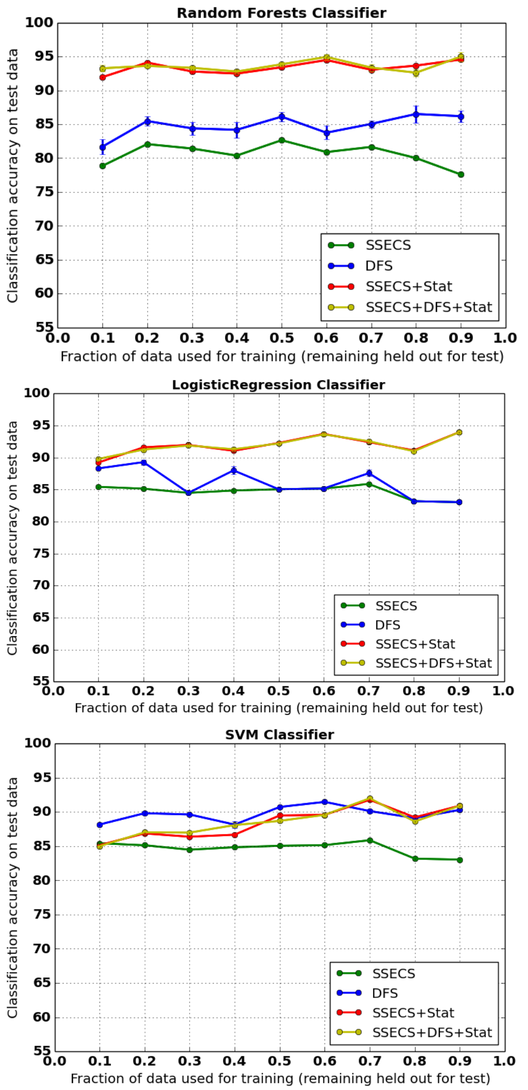

Figure 6. Illustrating the efficacy of the Distance from Shapelet (DFS) feature. Classifiers used are Random Forests (top), Logistic Regression (middle) and Support Vector Machine (bottom).

We plot some extracted shapelets (randomly picked) from our experiments in Figure 7. The figure shows four shapelets extracted from an experimental run. The four shapelets ($S_1 – S_4$) display a wide range of entropy patterns. The S_1 (top left) and S_3 (bottom left) shapelets are indicative of the malware class (plotted in red) while the S_2 (top right) and S_4 (bottom right) shapelets are indicative of the benign class (plotted in green). The lengths of the four shapelets are 12, 6, 25, and 7, and their distance thresholds for classification are 0.9382, 0.3481, 2.0527 and 1.7641. The shapelet-based decision tree consisting of these four shapelets is also shown.

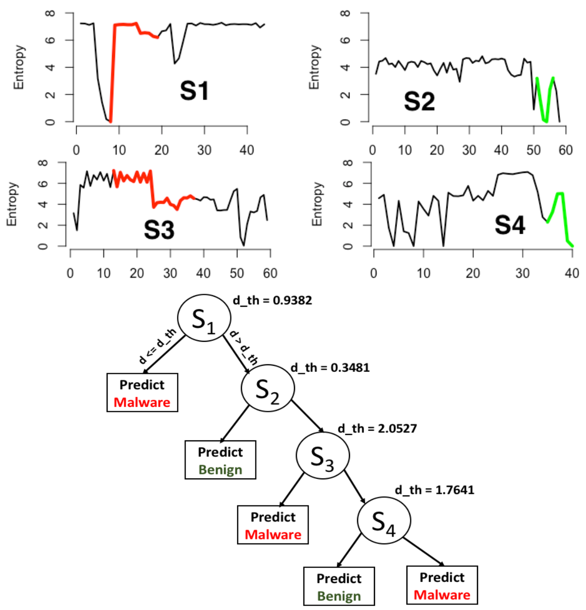

Figure 7. Shapelets extracted from an evaluation run (top) and the shapelet-based decision tree for classification of new instances (bottom)

7. Conclusions and future work

In this work, we motivated the application of time series shapelets for malware detection from executable files. Using entropy analysis and shapelet-based classification, we were able to quickly and accurately detect malware from new data. These shapelets can be used to perform root cause analysis by domain experts (who can interpret their meanings), to help direct the attention of machine learning algorithms (which process raw bytes of the computer program) in “interesting” places, to construct shapelet-related features in a larger pool of file features (strings, n-grams, etc.) for malware detection, or as part of an expert system that would pool its judgment along with other expert systems.

A future direction of research is to find an enhanced method for ranking the shapelets found and remove any false positives. Network communications typically deal with temporal data, and can serve as an exciting future use case. Mining timestamped log files for detecting attacks on a web server is another possible avenue for applying our work. Our results encourage further use of shapelet-based time series approaches to diverse computer security problems.

Acknowledgements

The authors thank Derek Soeder, Principal Researcher at Cylance Inc., for his contributions in numerous fruitful discussions about entropy analysis, the PE file format and malware detection.

Table 1. Classification accuracies when using the distance from shapelet as a feature on the J=5 dataset (best results highlighted in bold). DFS performs better than SSECS, and also improves accuracy of SSECS+Stat when added to it.

Fraction of data (*r*) used for training → (rest for testing)	***r=0.1***	***r=0.2***	***r=0.3***	***r=0.4***	***r=0.5***	***r=0.6***	***r=0.7***	***r=0.8***	***r=0.9***
Random Forests									
SSECS	78.98	81.59	81.41	81.38	82.38	81.36	81.43	79.37	80.0
DFS	82.14	85.52	84.44	84.43	85.92	83.44	85.49	86.76	86.61
SSECS+Stat	91.1	93.63	92.78	**93.08**	**94.04**	**95.1**	92.83	92.7	**96.36**
SSECS+Stat+DFS	**92.97**	**93.68**	**93.39**	92.74	93.93	95.01	**93.23**	**92.92**	95.33
Logistic Regression									
SSECS	85.42	85.13	84.48	84.84	85.04	85.15	85.86	83.17	83.03
DFS	88.3	89.28	84.48	87.99	85.04	85.15	87.55	83.17	83.03
SSECS+Stat	88.58	**91.58**	91.79	91.05	**92.27**	93.21	92.19	89.52	**93.94**
SSECS+Stat+DFS	**89.77**	91.28	**91.81**	**91.34**	92.24	**93.57**	**92.57**	**90.79**	**93.94**
SVM									
SSECS	85.42	85.13	84.48	84.84	85.04	85.15	85.86	83.17	83.03
DFS	**88.16**	**89.81**	**89.63**	**88.15**	**90.72**	**91.47**	90.13	89.11	90.3
SSECS+Stat	85.14	86.86	86.37	86.67	89.48	89.57	91.77	**89.21**	**90.91**
SSECS+Stat+DFS	85.0	87.01	86.98	88.08	88.69	89.57	**91.98**	88.6	**90.91**

8. References

[1] Supporting webpage for this paper at http://www-scf.usc.edu/~patri/hicss17.html

[2] Symantec Corporation, "2016 Internet Security Threat Report." Mountain View, CA, April 2016.

[3] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto. "Novel feature extraction, selection and fusion for effective malware family classification." *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy,* ACM, 2016.

[4] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab. "Zero-day malware detection based on supervised learning algorithms of API call signatures." *Proceedings of the Ninth Australasian Data Mining Conference* Volume 121. Australian Computer Society, Inc., 2011

[5] D. Baysa, R. M. Low, and M. Stamp. "Structural entropy and metamorphic malware." *Journal of Computer Virology and Hacking Techniques* 9.4 (2013): 179-192.

[6] D. Bilar. "Opcodes as predictor for malware." *International Journal of Electronic Security and Digital Forensics* 1.2 (2007): 156-168.

[7] L. Bilge, and T. Dumitras. "Before we knew it: an empirical study of zero-day attacks in the real world." *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[8] A. Davis, and M. Wolff. "Deep learning on disassembly data." *Black Hat USA*, 2015.

[9] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Vol. 1. Springer, Berlin: Springer series in statistics, 2001.

[10] S. L. Garfinkel. "Digital forensics research: The next 10 years." *Digital Investigation* 7 (2010): S64-S73.

[11] M. F. Ghalwash, V. Radosavljevic, and Z. Obradovic. "Extraction of interpretable multivariate patterns for early diagnostics." In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pp. 201-210. IEEE, 2013.

[12] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. "Learning time series shapelets." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.

[13] J. Z. Kolter, and M. A. Maloof. "Learning to detect malicious executables in the wild." *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004.

[14] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. "A symbolic representation of time series, with implications for streaming algorithms." *In Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2-11. ACM, 2003.

[15] J. Lines, Jason, L. M. Davis, J. Hills, and A. Bagnall. "A shapelet transform for time series classification." In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 289-297. ACM, 2012.

[16] R. Lyda, and J. Hamrock. “Using entropy analysis to find encrypted and packed malware.” *IEEE Security & Privacy* 2 (2007): 40-45.

[17] A. Moser, C. K. Andreas, and E. Kirda. “Limits of static analysis for malware detection.” *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007.

[18] A. Mueen, E. Keogh, and N. Young. “Logical-shapelets: an expressive primitive for time series classification.” In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1154-1162. ACM, 2011.

[19] P. O’Kane, S. Sezer, and K. McLaughlin. “Obfuscation: the hidden malware.” *IEEE Security & Privacy* 9, no. 5 (2011): 41-47.

[20] O. P. Patri, R. Kannan, A. V. Panangadan, and V. K. Prasanna. “Multivariate time series classification using inter-leaved shapelets.” *In NIPS 2015 Time Series Workshop. NIPS, 2015.*

[21] O. P. Patri, A. V. Panangadan, C. Chelmis, R. G. McKee, and V. K. Prasanna. “Predicting failures from oilfield sensor data using time series shapelets.” *In SPE Annual Technical Conference and Exhibition.* Society of Petroleum Engineers, 2014.

[22] O. P. Patri, A. V. Panangadan, C. Chelmis, R. G. McKee, and V. K. Prasanna. “Data Mining with Shapelets for Predicting Valve Failures in Gas Compressors.” *In SPE Western Regional Meeting*. Society of Petroleum Engineers, 2016.

[23] O. P. Patri, A. Panangadan, C. Chelmis, and V. K. Prasanna. “Extracting discriminative features for event-based electricity disaggregation.” *Proceedings of the 2nd Annual IEEE Conference on Technologies for Sustainability (SusTech)*. IEEE, 2014.

[24] O. P. Patri, A. B. Sharma, H. Chen, G. Jiang, A. V. Panangadan, and V. K. Prasanna. “Extracting discriminative shapelets from heterogeneous sensor data.” *In Big Data, 2014 IEEE International Conference on.* IEEE, 2014.

[25] F. Pedregosa, et al. “Scikit-learn: Machine learning in Python.” *The Journal of Machine Learning Research* 12 (2011): 2825-2830.

[26] M. Pietrek. “Peering inside the PE: a tour of the win32 (R) portable executable file format.” *Microsoft Systems Journal-US Edition* (1994): 15-38.

[27] T. Rakthanmanon, and E. Keogh. “Fast shapelets: A scalable algorithm for discovering time series shapelets.” *Proceedings of the 13th SIAM International Conference on Data Mining*, SDM, 2013.

[28] A. Renyi. “On measures of entropy and information.” *In Fourth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 547-561. 1961.

[29] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. “Data mining methods for detection of new malicious executables.” In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on* (pp. 38-49). IEEE, 2001.

[30] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer. “Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey.” *Information Security Technical Report* 14.1 (2009): 16-29.

[31] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq. “PE-Miner: Mining structural information to detect malicious executables in realtime.” *Recent advances in intrusion detection*. Springer Berlin Heidelberg, 2009.

[32] M. Siddiqui, M. C. Wang, and J. Lee. “A survey of data mining techniques for malware detection using file features.” *Proceedings of the 46th Annual Southeast Regional Conference on XX*. ACM, 2008.

[33] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo. “On the infeasibility of modeling polymorphic shellcode.” *Machine learning* 81, no. 2 (2010): 179-205.

[34] I. Sorokin. “Comparing files using structural entropy.” *Journal in Computer Virology* 7.4 (2011): 259-265.

[35] G. Tahan, L. Rokach, and Y. Shahar. “Mal-ID: Automatic malware detection using common segment analysis and meta-features.” *The Journal of Machine Learning Research* 13.1 (2012): 949-979.

[36] M. Wojnowicz, G. Chisholm, and M. Wolff. “Suspiciously Structured Entropy: Wavelet Decomposition of Software Entropy Reveals Symptoms of Malware in the Energy Spectrum.” *Florida Artificial Intelligence Research Society Conference*, 2016.

[37] M. Wojnowicz, G. Chisholm, M. Wolff, and X. Zhao. “Wavelet decomposition of software entropy reveals symptoms of malicious code.” *arXiv preprint* arXiv:1607.04950 (2016).

[38] Z. Xing, J. Pei, and E. Keogh. “A brief survey on sequence classification.” *ACM SIGKDD Explorations Newsletter* 12, no. 1 (2010): 40-48.

[39] L. Ye, and E. Keogh. “Time series shapelets: a novel technique that allows accurate, interpretable and fast classification.” *Data Mining and Knowledge Discovery*, 22(1-2):149–182, 2011.