Tensor Decompositions for Malware Classification: *Draft*

Tyler A. Simon and Rebecca M. Ward
Laboratory for Physical Sciences
University of Maryland
College Park, MD 20740

Abstract— **As an exploration into a novel malware identification and analysis technique, the ETTB++ tensor decomposition toolkit was used to examine goodware and malware files to see if it could both identify and cluster malware files. To this end, the following three principle experiments were conducted using ETTB++:**

- **Classifying file types;**
- **Distinguishing between malicious and benign PDFs;**
- **Identifying and clustering sophisticated exploit kits.**

This paper gives a brief primer on tensors and the ETTB++ tool, and then presents the methods and results from each of these experiments. Preliminary findings showed that the tensor decomposition tool was an effective means for distinguishing between file types, and that it has potential for identifying and clustering malicious files. Specifically, the major findings presented in this report are:

- **The tensor decomposition separated 81 out of 84 files into their respective file types by grouping like file types into the same components. This represented a successful demonstration of unsupervised clustering and suggested that the tool could also be used in conjunction with supervised learning techniques to identify the file types of unknown files at scale.**
- **Used in combination with a simple post-processing procedure, the tensor decomposition tool separated malicious and benign files of the same type with around 75% accuracy. Further investigation is needed to determine whether this is a true result or the artifact of a biased dataset.**
- **When used on a fused data set comprised of disparate data types with a naive decomposition, the tool did not generate meaningful results. A better understanding of how to craft good vectors and perform intelligent decomposition is needed.**
- **Used in combination with a pre-processing procedure based on subject matter expertise, the tensor decomposition tool clustered malicious files from sophisticated exploit kits with over 70% accuracy and separated malicious and benign files with around 75% accuracy.**

The results demonstrate that the tensor decomposition tool has utility for unsupervised learning and clustering, and that this utility is augmented when it is used in combination with other pre- and post-processing procedures and supervised learning techniques. The results also indicate that additional work is needed to understand the art of crafting good tensors and choosing an appropriate decomposition for different types of problems and data.

I. INTRODUCTION

A. Tensor decomposition methods

Here I need to put some background on Tensors. There is a good paper by Chang et. al.[1] that touches on the approach presented in this paper.

B. toolkit used during study

This section is not intended to give a thorough introduction to the ETTB++ toolkit; rather, it describes basic terminology and highlights the relevant features of the software necessary to understand the results presented in this paper. For a detailed description of the theory and operation of the software, see [2].

ETTB++ stores and processes data in a multi-dimensional array called a tensor. A tensor mode is simply a column in the multidimensional array. The toolkit performs matrix operations using a user specified algorithm, which performs an eigenvalue decompositions on this data structure. This process produces features called components which demonstrate various patterns in the initial data. The algorithms used for this study are ALS, which uses an alternating least squares fit of the data and an APR method which assumes the data is structured as a Poisson distribution. Both methods are investigated in this study.

The software is launched from the command line and can be interacted with through a GUI. It takes as input a delimited text file, referred to hereafter as a csv. The input text file contains different columns of data, which are the modes in the decomposition. Each row represents a different data point. Figure 1 shows a sample of the data used in the experiments in this paper. Here the bytes are used as the modes of the decomposition. The individual values in each row are referred to here as entries. The user selects the number of modes from the input file to be used in the decomposition, the type of decomposition (ALS or APR), and specifies the number of components into which the tensor should be decomposed. The resultant components are represented by component plots, which show the eigenvalue for each value in each mode. Each component has an associated weight, which describes how prominent the behavior described by that behavior ishigher weighted components describe more prominent behavioral patterns. As a processing step, the software assigns a unique integer

value to each unique entry in a mode. So for example in figure 1, for the byte0 mode, 50 would be assigned a value of 0, ff a value of 1, 89 a value of 2, etc. These mapped integer values are used for the decomposition and are retained in the results; thus, the values shown in the decomposition results correspond to this mapped value, NOT to the original value in the raw data. The reader should bear this in mind when interpreting the output of the tensor toolkit. The results presented in this paper are largely from the static component view and it is not immediately obvious which raw data value the results correspond to. For clarity and ease of interpretation, data labels were derived by referring back to the raw input data and are shown on many of the plots in this paper.

	A	B	C	D	E	F	G	H	I	J	K
1	byte0	byte1	byte2	byte3	byte4	byte5	byte6	byte7	byte8	byte9	byte10
2	50	4b	3	4	14	0	6	0	8	0	0
3	ff	d8	ff	e1	0f	fe	45	78	69	66	0
4	89	50	4e	47	0d	0a	1a	0a	0	0	0
5	50	4b	3	4	14	0	6	0	8	0	0
6	7b	5c	72	74	66	31	5c	61	64	65	66
7	50	4b	3	4	14	0	6	0	8	0	0
8	7b	5c	72	74	66	31	5c	61	64	65	66
9	25	50	44	46	2d	31	2e	35	0d	0a	25
10	50	4b	3	4	14	0	6	0	8	0	0
11	ff	d8	ff	e0	0	10	4a	46	49	46	0
12	89	50	4e	47	0d	0a	1a	0a	0	0	0
13	4d	5a	90	0	3	0	0	0	4	0	0
14	4d	5a	90	0	3	0	0	0	4	0	0
15	ff	d8	ff	e0	0	10	4a	46	49	46	0
16	89	50	4e	47	0d	0a	1a	0a	0	0	0
17	7b	5c	72	74	66	31	5c	61	64	65	66
18	7b	5c	72	74	66	31	5c	61	64	65	66
19	4d	5a	90	0	3	0	0	0	4	0	0
20	50	4b	3	4	14	0	6	0	8	0	0

Fig. 1. Sample of Header bytes for 20 files

II. File Classification

A. Experiment 1

A dataset with 83 files of different file types was collected. The file type breakdown is shown in Table I. A python script was used to extract bytes 0-93 and 128-192 from all the files in sample directory and to print these bytes to a csv. These bytes were selected because they are in the header of the file, which contains file type information, and because these specific bytes are the ones exploited by the Linux file command to identify the file type. The output csv file is comma delimited with a separate row entry for each file, with each byte in the file added as a separate column. The last column in each entry is the name of the file, which is not used in the tensor decomposition, but is used in the post-processing to validate results. A 10-component APR decomposition was run on the data, using only the first 8 bytes of the file as the modes.

III. Experiment 1 Results

The final fit for the decomposition was high at around 0.87, indicating a good fit. Of the 10 components generated, only 9 were really viable components: the weights ranged from 1.6-30 for the top 9 components, and the weight for the bottom component was on the order of 10-95, meaning it essentially

TABLE I
DATASET 1 FILES

Type	Details	Number of Files
pdf	v.1.5(5) v.1.4(3) v.1.6(1) v.1.7(1)	10
.docx		10
.pptx		10
.xlsx		10
jpg	JFIF(8) EXIF(2)	10
.rtf		10
.png		10
executables	PE32(gui)I386(5) PE32(console)I386(2) PE32+ x86_64(2) PE32+ x86_64(1)	10
others	.sh(1) .pl(1) .csv(1)	3

didn't exist and was only created because the decomposition was forced into 10 components. Using the software's interactive GUI and referring back to the original data, all of the components could be correlated with one of the file types. Table II summarizes the components, their weights, and the file types they describe. These results confirm that the tensor decomposition did successfully distinguish and cluster the different file types. Figure 2 shows the component plots labeled with the type of file they represent.

TABLE II
RESULTING DECOMPOSITION COMPONENT WEIGHTS

Component	Weight	File type
4	30	MS Office (.pptx, .docx, .xlsx)
2	10	PDF
6	10	rtf
7	10	png
8	8	JPG (JFIF standard)
1	2	JPG (EXIF standard)
3	8.45	PE
0	1.55	PE
5	1.55E-95	PE
9	3	others (.sh, .pl, .csv)

Several interesting findings resulted from this work. One unexpected result was that the weight of the component was equal to the number of files of that type in the original data set. For example, the component that described the MS Office suite of products was component 4, which had a weight of 30. This corresponded directly to the 30 MS Offices files in the dataset. Another unexpected result was that the decomposition broke the executables into three different components, even though all ten files were identical in the first eight bytes. This curious result prompted the question as to whether looking at more than the first eight bytes would allow the decomposition to further distinguish between the different types of executables (e.g. PE32 vs. PE32+). The decomposition distinguished between JFIF and

EXIF standard JPGs, which was a file characteristic unknown to the authors until examining the results. It also grouped all ten PDF files into one component, even though they varied in the eight byte. This byte described which version of Adobe was used to create the PDF. Figure 3 shows this variation and how it is called out by the decomposition. The fact that all ten PDFs were grouped together despite variations in the final byte reinforces the utility of the tensor decomposition for identifying latent behavioral patterns.

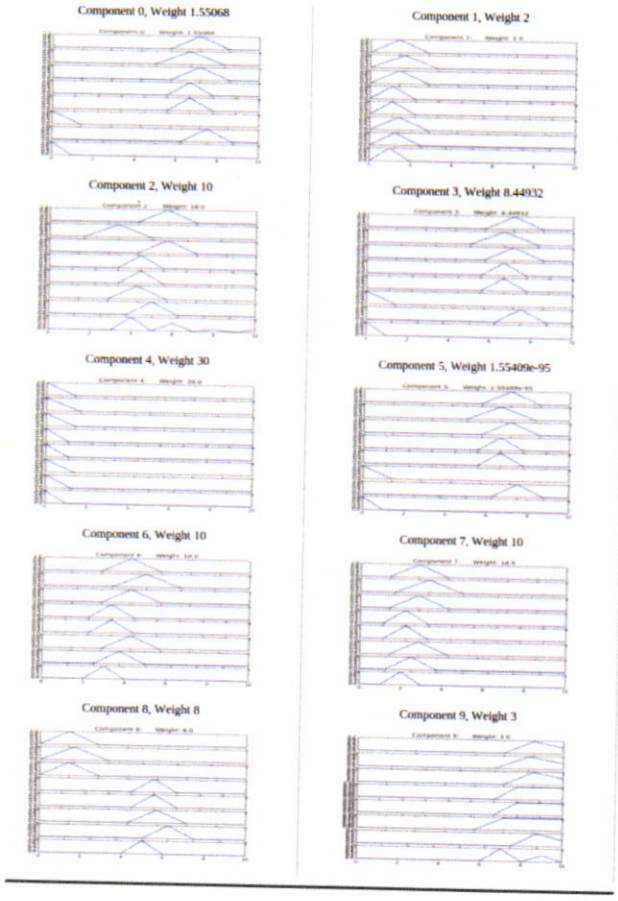

Fig. 2. Component graphs from APR decomposition for file type classification

In figure 3 The different values represent different versions of Adobe Acrobat used to create the file.

IV. Identifying Malicious PDFs

A. Experimental Methods

Once it was established that the file type could be distinguished using a tensor decomposition, the next experiment was crafted to determine whether or not the software could aid in distinguishing between malicious and benign files of the same type. The premise in this case was that malicious files may contain information in their headers that benign files do not, such as information related to code obfuscation or instructions for executing code in a non-standard order. In order to test this hypothesis, a collection of 109 PDF files

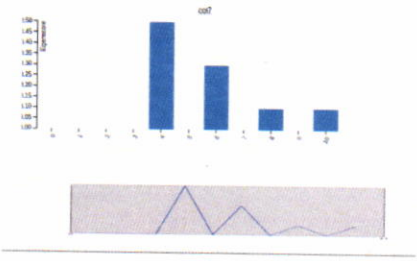

Fig. 3. The eighth mode of component 2, corresponding to eighth byte of PDF files.

was assembled, including 59 benign PDFs and 50 malicious PDFs. The same script that was used in the classification experiment was used here to extract targeted header information from all the files and save this information to a formatted csv file.

Several different decompositions were run on the data set, varying the number of modes used, the number of components, and the decomposition type. The results discussed here are for a 10-component ALS decomposition using 17 modes (bytes 15-32 of the header information). Note this limited number of components and modes represents an overly simplified experiment and was used here for ease and clarity of analysis. Once the components had been generated and correlated with the original dataset, an additional post-processing step was undertaken to see if the decomposition successfully discriminated between malicious and benign files. A feature vector was developed from each component based on the entry in each mode with the highest eigenvalue. In practice, this meant for each component, looking at each mode and pulling out the highest peak, and identifying the byte value to which the peak corresponded. Figure 4 shows the component plots for components 1 and 9 and gives the corresponding feature vectors that were generated for these components. The feature vectors were then used to perform clustering on the data, which is described below in the results section.

B. Decomposition with n-grams

A second analysis performed on the PDF dataset was inspired by a machine learning-based malware identification tool developed at LPS. The tool uses an n-gramming approach on labeled data to generate features for the tool to learn on. A file is parsed into 6-byte n-grams using a sliding window, and all of the n-grams from each file are stored. After all the malicious and benign files have been n-grammed, a malice score is assigned to each n-gram based on its relative prevalence in malicious and benign files; that is, n-grams that occur primarily in malicious files and rarely in benign files are assigned a high malice score, and vice versa. This same approach was used herethe PDF files were n-grammed, and the top 10 n-grams with the highest malice scores were retained in a keep list. A 10-mode binary vector was then generated for each file: for each n-gram in the keep list, if it occurred in the file, a value of 1 was assigned; if it did not, a value of 0 was assigned. These

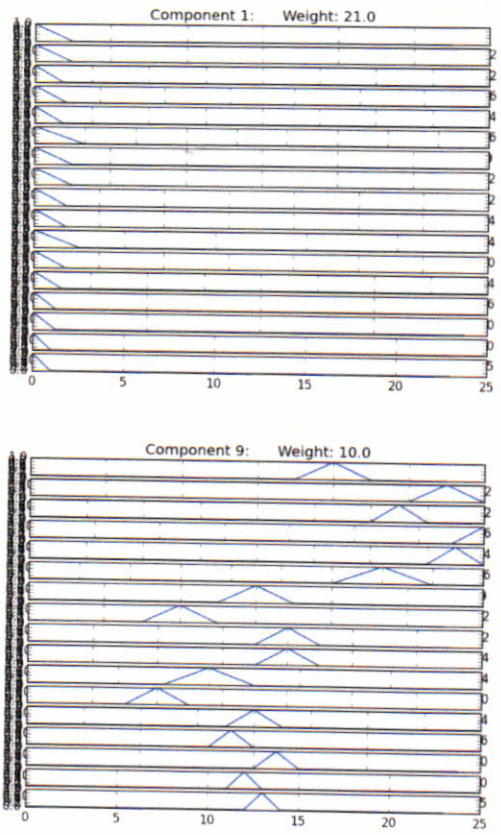

Fig. 4. Components 1 and 9 of 10-component ALS decomposition on malicious and benign PDFs, and the feature vectors developed from those components

binary vectors were then fused with the header-byte vectors described in the previous section into one data file. The data file had 109 rows, each representing a different PDF file, with 169 modesthe first 10 modes were the binary n-gram vectors, the next 156 modes were the header byte values, and the last mode was the file name, used in analysis of the results. This file was used as input to the tensor toolkit, and several decompositions were run with 27 modes (the binary n-gram data and bytes 15-32). This experiment represents an augmentation of the previous experiment, as the data used in the previous experiment has been fused with the n-gram data, which tests an important feature of the decomposition tool; namely, the ability to find behavioral patterns in disparate data.

V. PDF Results

A. Decomposition with header information

Using bytes 15-32 of the files as the modes in an ALS decomposition, 10 components were generated to describe the PDF dataset comprised of malicious and benign PDFs. As was the case in the file type classification experiment, all of the components could be correlated with the elements in the original dataset that they described. Each component described either a set of good files or a set of bad files, which was a promising result, indicating that malicious files had been grouped together and benign files had been grouped together. Six of the components represented benign files, and the other four represented malicious files.

Feature vectors were generated from each component, as described in section 2.1. Visual inspection of these 17-element feature vectors suggested that the vectors that described good files had more bytes in common with each other than they did vectors that described bad files. To test this hypothesis, a naive similarity metric was employed to quantify how similar two vectors were: if the same byte value occurred in the same position for both vectors, a 1 was assigned; otherwise, a 0 was assigned. The sum of the comparison values was divided by the total elements in the vector (17), and the resultant value ranged from 0 (no bytes in common) to 1 (all bytes the same). This value was used as the similarity metric. The most heavily weighted component associated with good files, component 9, was used as the center of the good feature vectors, and the most heavily weighted component associated with bad files, component 1, was used as the center of the bad feature vectors. Each vector was compared to the good vector and the bad vector, and a goodness and badness similarity were computed. Thus each component in the decomposition was associated with a coordinate pair (x,y), where x was the goodness similarity and y was the badness similarity. These pairs were plotted to see how well the good and bad components clustered. Figure 5 shows this plot, with the bad components represented by red points, and the good components represented by blue points. Note that the true goodness or badness of the file was determined by referring back to the original data, so the red and blue dots show ground truth. The position in the xy-plane shows the computed file maliciousness based on the results of the tensor decomposition and similarity analysis described above.

The preliminary results of this analysis are promising, though additional work is needed to determine whether they are meaningful or largely an artifact of the dataset. Figure 5 shows that all of the bad components have a non-zero badness similarity score, and only one of them has a non-zero goodness similarity score. The goodness similarity scores were not as discriminative, however, as half of the good components had non-zero scores, and half had scores of zero. Nonetheless, the components are nearly linearly separable, as illustrated by the dashed blue line in Figure 5. Three out of the four bad components fall above this line, and four out of the five good components fall below it, showing that this simple analysis has a discrimination accuracy of over 75%. Further investigation is needed to determine whether the toolkit is discriminating these components based on truly malicious characteristics, or if there is some other latent difference that is being picked up by the tensor decomposition (e.g. different Adobe versions being used to create the good versus bad files because one dataset is older than the other).

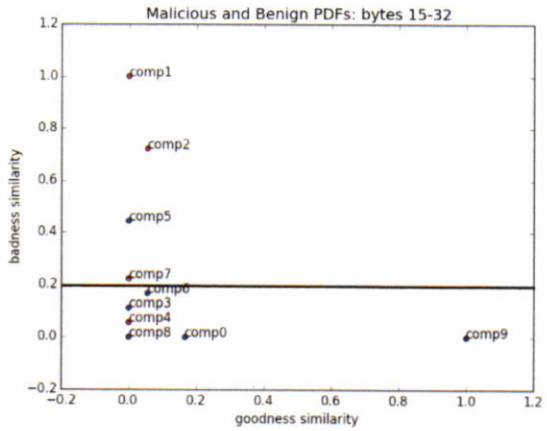

Fig. 5. Results of tensor decomposition on good and bad pdfs. Ground truth is shown in red (malicious files) and blue (benign files), and the maliciousness determined by the analysis is shown by the position of the points.

B. decomposition with n-grams

Several decompositions with varying parameters were run on the hybrid n-gram header bytes dataset, including both APR and ALS decompositions. The APR decomposition did not produce viable results, as it had a final fit equal to zero, and all of the component weights were small (less than 10-11). ALS decompositions with 3, 10, and 109 components were run. The 10- and 109-component runs both generated an error, as they resulted in a singular matrix so the tensor could not be fit to the specified number of components using double precision arithmetic. When this occurs, the tool throws a prompt with this error, and explains that this may be occurring because the data cannot be compressed in a meaningful way. After these failed trials, a 3-component ALS decomposition was successfully run, with a final fit of 0.15. All three components generated had the same weight, and they also all had identical results for the first ten modes, which were the modes that corresponded to the n-gram data. This result suggests that only differences in the byte data were being identified by the decomposition, meaning no value was being extracted from the n-gram data. Here the combination of n-gram and byte data did not generate useful results. This experiment underscores the importance of crafting a good tensor and performing an intelligent decomposition to generate meaningful results. While this tool is touted as having utility for fusing disparate data sets, the results of this experiment show that such fusion requires some effort on behalf of the user or an external analyst, we regard this as an area that would benefit from further exploration.

VI. Sophisticated Exploit Kit Analysis

A. Experimental Methods

Three sophisticated exploit kits were downloaded from the Internet for analysis. Two of the exploits were comprised of 11 executables each, and the third of only one executable. Table III summarizes the breakdown of the malicious files in each family. These 23 malicious files were collected in a dataset with 20 benign files, and a byte extraction script was run on all the files to extract the first 1024 bytes. As described above, these bytes were assembled in a csv, which was used as input data for the toolkit. Both APR and ALS analyses were run on the data with varying numbers of modes, but none of these runs yielded meaningful or interpret able results.

TABLE III
Summary of files on sophisticated exploit data set

File Family	Number of Files
Exploit 1	1
Exploit 2	11
Exploit 3	11
Benign Executables	20

B. Decomposition on file feature data

While the byte data described above is an approximate representation of the raw file, it does not capture characteristics of the entire file and makes for difficult interpretation due to the large number of modes. A second experiment was designed to use pre-processed data that describes features of the files that have potential relevance to the maliciousness of a file. The data processing technique used here was taken from the malware detection literature, specifically a 2008 paper by Perdisci et al. on identifying packed executables to aid in malware detection [3]. The paper presents an analysis tool that scans an executable and extracts nine file features which can be used as input vectors to supervised learning techniques to detected packed executables. In this work the vectors were assembled into a dataset to be used with an unsupervised learning technique, the tensor toolkit. A description of nine features extracted from executables is listed below. This analysis was performed on the same dataset presented above, with 23 sophisticated malware files from three families and 20 benign executables.

- Number of standard sections
- Number of non-standard sections
- Number of Executable sections
- Number of Readable/ Writable/ Executable sections
- Number of entries in the IAT
- Entropy of the PE header
- Entropy of the code section
- Entropy of the data section
- Entropy of the entire PE file

C. Results

A 4-component APR decomposition was performed on the 43-file dataset. Four components were chosen because the dataset was comprised of three families of malware and a set of goodware, so the intent was for each component to describe one family of files. Table IV shows the results of the decomposition, which illustrate that the tensor toolkit was, for the most part, able to separate malware and goodware into

different components. Additionally, it did group malicious files from the same family together. The first component captured the single file in Exploit1, as well as 8 out of the 11 files in Exploit2. The second component found 8 out of the 11 components in Exploit3, and the third and fourth components together described 15 out of the 20 benign files. Considering the first two components the bad components and the second two the good components, this analysis yielded a malware detection accuracy of about 74%. This result is impressive given that the malicious files analyzed in this experiment are from sophisticated exploit kits that are designed to mimic benign files as closely as possible. The results of this experiment demonstrate the utility of coupling the unsupervised learning capability of the tensor toolkit with intelligent data processing to find relationships in the data that are difficult to find with data processing alone.

TABLE IV
RESULTS FROM THE APR DECOMPOSITION ON THE FILE FEATURE VECTORS

Component	Files Described by component	Number of files
1	Exploit1	1
	Exploit2	8
	Benign	5
2	Exploit3	8
	Benign	1
3	Benign	6
4	Benign	9
	Exploit2	2
	Exploit3	2

VII. CONCLUSIONS

Three principle experiments were conducted using the ETTB++ tensor decomposition toolkit to analyze malicious and benign files: 1) classification of file types; 2) discrimination between malicious and benign files of the same type; and 3) grouping families of malicious files together. The results of these experiments showed that the toolkit could successfully be used to classify file types, and that it has potential for discriminating between malicious and benign files and for grouping families of malicious files together. In addition to these immediate results, these experiments showed that a carefully crafted dataset and tensor, as well as a thoughtful decomposition are necessary to produce meaningful results. To that end, future work in this vein should focus on gaining a better understanding of how the decompositions work, which type of decomposition is appropriate for what type of data, and how tensors should be formed for different types of problems. Additionally, the results of experiments two and three demonstrated the power of coupling the tensor decomposition software with other pre- and post-processing data analysis techniques to find relationships in the data that neither the tensor toolkit nor the data processing techniques alone are able to find. This is another area that warrants further exploration, particularly understanding how the decomposed components can be used to identify clusters of data that exhibit similar behavior.

ACKNOWLEDGMENTS

The authors appreciate the help of Albert Scalo, Mark Mclean, and Chris Krieger at the LPS.

REFERENCES

[1] T.-Y. Chang, W.-Y. Lai, T.-R. Hsiang, and C.-H. Mao, "Detecting malware in malicious virtual machines using tensor analysis techniques," in *Frontiers in Artificial Intelligence and Applications*, vol. 274, 2014.

[2] T. D. Plantenga and T. G. Kolda, "C++ tensor toolbox user manual (v 1.0) c++ tensor toolbox user manual (v 1.0)," 2012.

[3] R. Perdisci, A. Lanzi, and W. Lee, "Classification of packed executables for accurate computer virus detection," *Pattern Recogn. Lett.*, vol. 29, no. 14, pp. 1941–1946, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2008.06.016