# Malicious traffic detection using traffic fingerprint

## Abstract

We consider the problem of detecting malicious traffic in high bandwidth links. With the ever increasing bandwidth and traffic, deep packet inspection interferes with throughput and becomes computationally demanding. We developed a learning algorithm based on the well-known universal compression algorithm Lempel-Ziv 78 [1]. We built a proof-of-concept application that emulates a real-world situation and attempts to identify malware. Our algorithm builds a traffic fingerprint for well-known malware using only the time difference between packets. The algorithm then compares the fingerprint against unknown traffic.

We study the effectiveness of our method with real-world malicious traffic.

## Introduction

Cyber-attacks are an attempt to damage, disrupt, or gain unauthorized access to a computer, computing systems or a network. Cyber-attacks can affect a wide range of domains and system and potentially cause tangible damage to lives in case of SCADA networks.

Malware is a piece of computer software that uses vulnerabilities in computer hardware and software in order to alter the state or function of computers and computer networks without permission (explicit or implicit). Modern malwares depends on communication networks in order to receive commands, coordinate attacks (DDoS), relay information to the attacker and infect new targets.

Detecting malicious traffic in high bandwidth links is a challenging and complicated task. One of more prevalent solutions is deep packet inspection (DPI). DPI scans the entire packet stream, and can be used to identify malware communication in the data section of the packet. The main drawbacks of this approach are the computational power needed to classify the traffic, as well as the difficulty of inspecting encrypted packets.

A different solution is feature extraction, which attempts to overcome the disadvantage of deep packet inspection by extracting a limited number of features from the packet. When performing an analysis over large amount of traffic data and malwares, deciding which features to extract requires a large amount of memory and computation power.

Our research focuses on a different approach for traffic classification, known as traffic fingerprint. This method overcomes some of the disadvantages of the methods described. Our solution examines only the time difference between packets.

The learning algorithm we represent is based on the well-known universal compression algorithm Lempel-Ziv 78 [1]. We create a traffic fingerprint using the time differences from malware communication. We represent malware communication events as discrete sequences over small finite alphabet. This sequence is then used for building a Lempel-Ziv 78 tree with a probabilistic prediction model [2]. This modified tree is used as fingerprint for each specific malware and represents the malware behaviour. A similar approach was used when attempting to identify a unique user typing on a computer [3].

This approach enables us to make fast and accurate decisions without the need for packet analysis.

## Preliminaries

### Lempel Ziv 78

In 1978, Avraham Lempel and Jacob Ziv presented their algorithm for variable-rate compression [1] (LZ78). Their dictionary-based algorithm has been used extensively for compressing many different file types, from images to text and audio.

The LZ78 algorithm is a universal prediction, one pass algorithm. It builds a weighted tree from sequences of a finite alphabet.

The LZ78 tree holds a ‘dictionary’ of phrases parsed from the input text (training) and is constructed incrementally as follows:

At the beginning, the dictionary is empty. During each step of the algorithm, the smallest prefix of consecutive symbols not yet seen is added to the dictionary. As such, each phrase is unique in the dictionary and it extends a previously seen phrase by one symbol.

For example, the string '*abbacbaccbcabb*' is parsed into the following dictionary entries *a; b; ba; c; bac; cb; ca; bb*.

Since each phrase extends a previously seen phrase, we can order them in a tree, as described in [4]. In [2] the authors proposed a method for prediction of the next outcome of a sequence using the aforementioned LZ78 tree, by assigning conditional probabilities to each event.

In [3], an expanded LZ78 tree is used in order to identify the user typing on a computer keyboard. The authors suggested an expansion of the LZ78 tree seen in [2] by using input shifting and back-shift parsing, in order to rectify ‘noisy’ statistics caused by small training sets.

The idea to apply the LZ78 tree prediction method to malware detection was first suggested in [5]. Their idea included using vector quantization on packet time differences in order to predict whether unknown traffic is malicious or not.

### Malware Traffic

Malware (contraction of malicious software) is any piece of software intended to gather information, cause damage or infiltrate a target computer without permission.

In the past, most malware was in the form of viruses and worms, and was usually distributed physically. In recent years, with the spread of computer networks in general and the internet in particular, most malware now utilizes the internet or a local network for spreading and coordinating actions.

When a virus coordinates actions, it is known as a botnet. Such actions might include distributed attacks (DDoS), personal data gathering, e-mail spam, etc. A botnet usually has a command and control (C&C) channel that it utilizes for coordinating such attacks.

The C&C channel is usually obscured by impersonating a well-known protocol to some extent, such as HTTP port 80, HTTPS port 443, IRC ports, etc.

The traffic generated by the malware when communicating through the C&C channel tends to remain consistent [6].

Let’s examine the Cryptolocker ransomware as a test case. Cryptolocker is a ransomware trojan which infects Windows based PCs. Usually, the attack begins as an e-mail attachment, after which the Trojan is installed on the PC. When activated, the Trojan encrypts a selection of files on the computer using RSA public-key cryptography with the private key stored on a remote server. The user is then given an ultimatum to pay bitcoins in order to receive the decryption key. If no payment is made by the deadline, the Trojan threatens to erase the key from the server and keep all of the data encrypted.

The virus therefore has an interesting network traffic profile. It first arrives via e-mail. Then, once installed it begins looking for a server. First, it attempts to access a hard-coded IP 184.164.136.134. If that fails, it generates a pseudo-random URL based on the time of day. This rule is known and allows the operator of the Trojan to pre-register the pseudo-random domain names. [7]

Once a suitable command and control server has been found, the malware will start to communicate through regular HTTP POST requests, albeit only as a wrapper for RSA encrypted data.

[להכניס תמונה של רשימת השיחות]

[להכניס תוכן של הפקטה]

This behaviour is for the most part constant and was identified in several captures on different days and different machines.

## Identifying traffic based on the LZ78 fingerprint

The core of our application is the LZ78 based fingerprint. Each fingerprint represents the behaviour of one malware capture.

We must first describe a method for transformation of the network behaviour into a parsable input sequence which is quantized using a vector-quantization clustering algorithm to be fed into the LZ78 tree creation algorithm.

### Representation via quantized packet time-difference

The packet time difference is the time elapsed between to packet arrival/departure events in the same flow. We will define the time difference between packets as.

Each stream of lengthis transformed into a sequence of time differences. Because  is in an infinite range, we wish to reduce the number of time differentials and smooth them over. Thus, we perform a vector quantization using K-Means clustering. This enables us to use fewer symbols in our learning phase which reduces variance.

#### K-means clustering and K-means++ quantization scheme

In order to quantize the packet time differences into a string parsable by the LZ78 tree-building algorithm, we used the K-Means++ algorithm.

K-Means clustering, the basis for the K-Means++ algorithm is commonly used to partition a data-set into k groups by selecting k clusters and then refining them iteratively. Since the algorithm is at its base NP-Hard, several algorithms, such as Lloyd-Max are commonly used to converge to an optimal result more quickly.

K-Means++ [4]is an algorithm for choosing the initial seed values of the K-Means clustering algorithm first proposed in 2007 by D. Arthur and S. Vassilvitskii. It is an approximation algorithm for the NP-Hard K-Means problem, which avoids the sometimes poor clustering found by the Lloyd-Max algorithm.

In our program, the input for K-Means++ is an observation vector which is all time differences seen during the extraction process described above.  
The output for K-Means++ is a list of centroid values.

In order to perform quantization for each malware capture, we first need to set decision boundaries. The decision boundaries are set halfway between every two centroids.

[להכניס איור מהדו"ח מכין כזה]

### Classification Methods

#### KL Distance

Kullback–Leibler divergence is a metric for measuring the distance between two probability measures, defined as 

In our program, each test is composed of two probability models, consisting of discrete probabilities for the fingerprint and the raw capture data. When comparing the two trees on a node-by-node basis - there could be a case where =0.

To counter this case, we implemented a modified version we call *Smoothed KL Distance*.

The algorithm runs on two trees: is the fingerprint, while  is the unknown capture.

sum:=0

For each node *i* in :

1. Get probability of the node 
2. If the exact node exists in :
   1. Assign with that probability.
   2. Else:
      1. Get probability of closest matching node (lexicographically)
      2. Assign probability to  and multiply by very small.
3. Add  to sum.

Return sum.

[לסדר את האמא של האלגוריתם הזה]

#### Hamming Loss and Log Loss

The two techniques Log loss [6] and Hamming loss [7] [8] are functions that represent the cost or value of an event, or an error metric.

The error metric is used because we wish to predict the time difference for the next packet, based on the context of the time differences we have seen so far.

Since we have already assigned probabilities for each event, these will be the probabilities that will feature in the loss functions.





In Log loss, the use of the log function on the probability causes extreme punishment for being confident about a wrong prediction.

In Hamming loss, any mismatch between the prediction and the real value will cause a punishment of 1.

### Program structure and algorithm

We built a proof-of-concept application using Python due to the large amount of add-on packages for parsing input files and numerous mathematic packages.

Our program is split into two operating phases: a training phase and a testing phase.

### Training phase

The pre-cleaned malware captures are processed to extract packet time differences.  
These time differences are placed in a vector which is then quantized into a user-specified amount of values (centroids) using the K-Means++ algorithm.

The quantized string is then used to build a LZ78 tree based on the algorithm presented previously. This represents our malware fingerprint. (This action is performed for each malware capture separately.)

The fingerprints are stored in a local malware fingerprint database for easy access.

The algorithm:

1. Scan all malware traffic capture files. Extract the time differences into a single observation vector .
2. Run K-Means++ to obtain centroids given the observation vector. Place centroids in.
3. For each malware traffic capture file:
   1. Extract time differences for this specific file into vector 
   2. Apply quantization transformation:



* 1. Map each quantized value into a corresponding letter from the Latin alphabet (a,b,c,…).
  2. Feed the quantized vector  as a string into the LZ78 tree generation algorithm.
  3. Insert the tree as a fingerprint into the fingerprint database.

### Testing Phase

In the testing phase, an unknown capture is to be assigned a score in comparison with our fingerprint database. The lower the score, the better the match.

A process identical to the training phase is performed on the unknown capture to be tested, with the exception of (3.e) – the capture is not placed in the database.

The unknown capture is then compared against the known malware database fingerprint by fingerprint, using three different algorithms – Smoothed KL-Distance, Log Loss and Hamming Loss.

Each of these algorithms returns a numerical value for each pair of fingerprints. These values will be used to classify the capture as malware or malware free.



### Data collection

Our goal was to test our system with real-world malware. In order to obtain captures, we cooperated with CYREN, which gave us access to their APT lab and database.

### Cleaning the captures

The captures obtained were very long and contained a lot of irrelevant traffic.

In order to clean the captures, we applied the same Wireshark display rule and exported a new, clean PCAP file.

The rule was intended to keep only TCP and DNS IPv4 traffic – essentially limiting the capture to WAN traffic only.

### Packet time difference extraction

Defined for every packet *i* such that 

### Building the fingerprint

In order to build a LZ78 tree, first we must generate a dictionary containing all unique entries from the quantized string.

For example, the string '*abbacbaccbcabb*' is parsed into the following dictionary entries *a; b; ba; c; bac; cb; ca; bb*.

A tree is then built based on the dictionary entries, in such a way that each entry that is a suffix of an existing tree node is inserted as its descendant. Notable exceptions are single letters which are inserted as descendants of the root.

After the tree is built, our algorithm assigns uniform probabilities from the bottom up, summing up the probabilities from the immediate descendants. This is based on the work shown by [3] and [2].



Figure 2 - Fingerprint tree

Figure 2 above is an example of a LZ78 tree generated for some malware traffic. The conditional probability is easy to calculate from this tree. For example, probability of event ‘ac’ given ‘a’ is

## Test results

For testing our application, we first cleaned 19 different captures of five main viruses: Renovator, Hesperbor, Darkcomet, Cryptlocker and Bladabindi.

These malware captures were recorded at different parts of the day on different computers. Some even contain slightly different behaviour, due to different set-ups and time of day.

The parameters we used in our program set the number of centroids to be produced to 25 and the threshold value to 3.0.

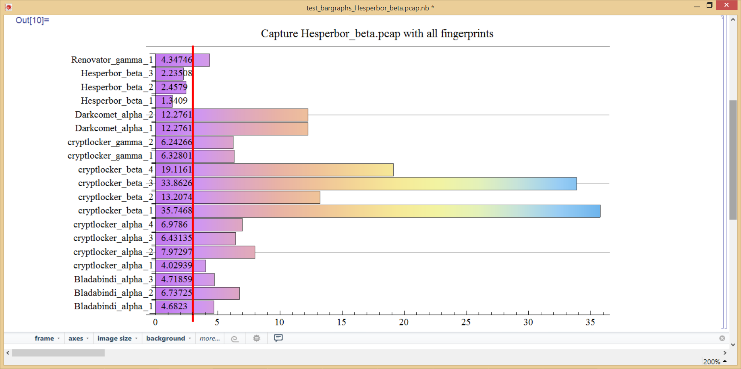


Figure 4 - Capture contaning Hesperbor behaviour

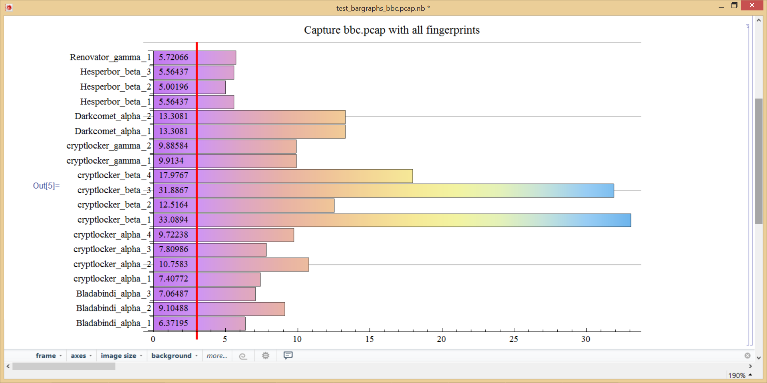


Figure 5 - Capture containing no known malware

In Figure 4, a virus called Hesperbor is hiding inside a ‘dirty’ capture containing a lot of varied traffic (surrounding and interspersed). The capture fingerprint was compared against the entire database. The values assigned represent the KL-distance between the capture fingerprint and the malware fingerprint. The greater the number, the more substantial the distance. The vertical line represents the empirically derived threshold.

If the value assigned is below the threshold then we can decide with good confidence what malware the capture contains.

In Figure 5 we took a clean capture of access to bbc.co.uk. We can see that none of our virus fingerprints were identified in this capture (all values above 3.0).

### ROC graph

Figure 6 - ROC graph showing testing behaviour for different malware/software fingerprints

This ROC graph shows the characteristic results with a few malware fingerprints and one non-malware protocol.

# Works Cited

|  |  |
| --- | --- |
| [1] | A. Lempel and J. Ziv, "Compression of individual sequences via variable-rate coding," *IEEE Transmissions on Information Theory,* pp. 530-536, September 1978. |
| [2] | M. Feder, N. Merhav and M. Gutman, "Universal prediction of individual sequences," *IEEE Transactions on Information Theory,* vol. 38, no. 4, pp. 1258-1270, July 1992. |
| [3] | M. Nisenson, I. Yariv, R. El-Yaniv and R. Meir, "Towards Behaviometric Security Systems: Learning to Identify a Typist," *Knowledge Discovery in Databases: PKDD 2003, Lecture Notes in Computer Science,* vol. 2838, pp. 363-374, 2003. |
| [4] | G. J. Langdon, "A note on the Ziv - Lempel model for compressing individual sequences (Corresp.)," *IEEE Transactions on Information Theory,* vol. 29, no. 2, pp. 284-287, 1983. |
| [5] | A. Cohen, S. Dolev, N. Gilboa and G. Leshem, "Anomaly Detection, Dependence Analysis and," Beersheba, 2012. |
| [6] | N. Villeneuve and J. Bennett, "Detecting APT Activity with Network Traffic Analysis," 2012. |
| [7] | EMSISoft Blog, "CryptoLocker – a new ransomware variant," EMSISoft Blog, 10 September 2013. [Online]. Available: http://blog.emsisoft.com/2013/09/10/cryptolocker-a-new-ransomware-variant/. [Accessed 16 August 2014]. |
| [8] | D. Arthur and S. Vassilvitskii, "k-means+: The advantages of careful seeding," *Proceedings of the eighteens annual ACM-SIAM symposium on Discrete algorithms,* pp. 1027-1035, 2007. |
| [9] | C. M. Bishop, Pattern Recognition and Machine Learning, M. Jordan, Ed., Cambridge: Springer, 2006, p. 209. |
| [10] | G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining,* vol. 3, no. 3, pp. 1-13, 2007. |
| [11] | R. W. Hamming, "Error detecting and error correcting codes," *Bell Systems technical journal,* vol. 29, no. 2, pp. 147-160, 1950. |
| [12] | S. Kullback and R. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics,* vol. 22, no. 1, pp. 79-86, 1951. |