Malicious traffic detection using traffic fingerprint

Arnon Shimoni and Shachar Barhom
Academic Advisor: Dr. Asaf Cohen

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.

Index Terms

Anomaly Detection, Botnets, Command and control channels, Lempel-Ziv, Universal Compression, Probability assignment

1 Introduction

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

Malware[1] 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 approach is feature extraction, which attempts to overcome the disadvantage of DPI by extracting a limited number of features from the packet. When performing an analysis over fast data links for an ever growing list of malwares, deciding which features to extract requires copious amounts of memory and computational power.

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

The learning algorithm represented is based on the well-known universal compression algorithm Lempel-Ziv 78 [1]. A traffic fingerprint is created using the time differences from malware communication. Malware communication events are represented 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.

2 Preliminaries

2.1 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

A. Cohen, A. Shimoni and S. Barhom are with the Department of Communication System Engineering, Ben-Gurion University, Beer-Sheva, 84105, Israel. E-mails: {coasaf,arnons,barhoms}@post.bgu.ac.il

1. Contraction of **mal**icious soft**ware**

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: Initially, 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* (See example in Figure 1)

Since each phrase extends a previously seen phrase, they can be ordered 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

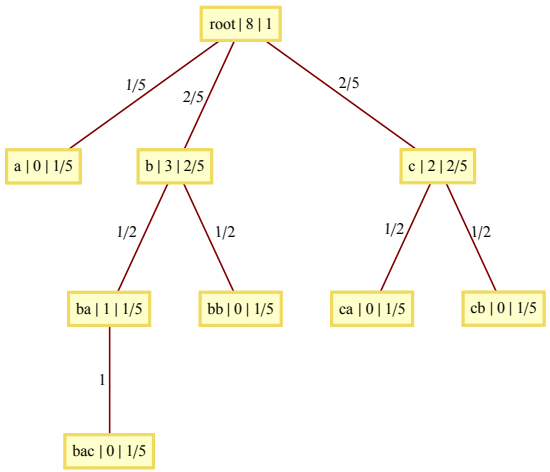


Fig. 1: Sample LZ78 tree generated from the string *'abbacbaccbcabb'*

user typing on a computer keyboard. The authors suggested an expansion of the LZ78 tree seen in [4] 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 - based on pre-trained sequences.

2.2 Malware Traffic

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 (like DDoS[2]), personal data gathering, e-mail spam, etc. A botnet usually has a C&C[3] 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].

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.

Examining the Cryptolocker ransomware as a test case, an interesting network traffic profile is revealed. It first arrives via e-mail or other method. 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 C&C server has been found, the malware will start to communicate through regular HTTP POST requests (See figure 2), albeit only as a wrapper for RSA encrypted data (See figure 3). This behaviour is for the most part constant and was identified in several captures on different times and different machines.

2. Distributed denial of service
3. Command and Control

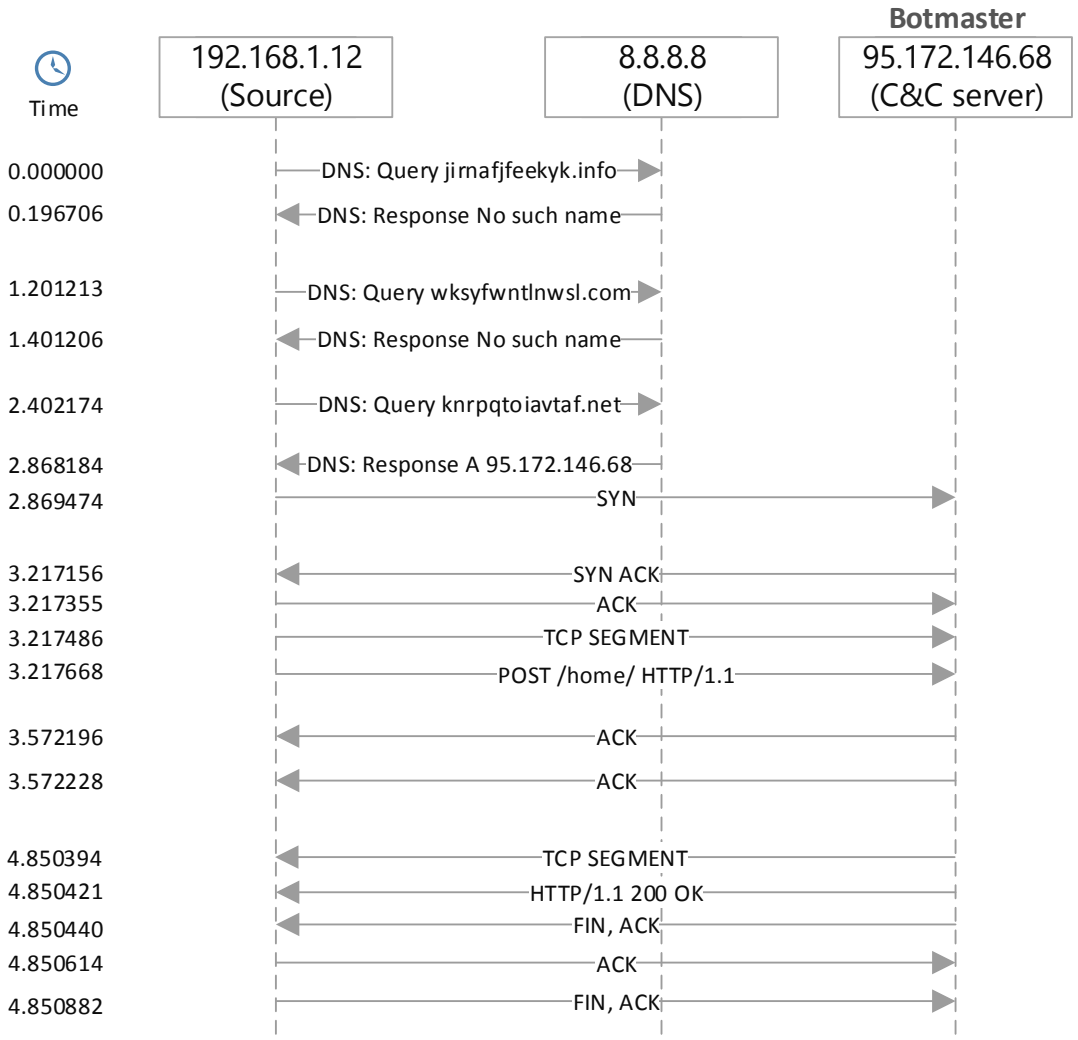


Fig. 2: Cryptolocker sample conversation list

```
POST /home/ HTTP/1.1
Cache-Control: no-cache
Connection: Close
Pragma: no-cache
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729;
        .NET CLR 3.0.30729; Media Center PC 6.0)
Content-Length: 192
Host: nrpqtoiavtaf.net

Z).h.../.=.ce:X.Y....y..:Qt.....M......".....
.k.k..tj..cT-.G].6/..s...g..J.........h....?!.9G8....=..G.g..P<U.n.........rM...\o.K?./M|......L.d).-.......E...1.i..o.i.Dc
        ..g.'..b.5Z....E......v.z
```

Fig. 3: Cryptolocker outgoing TCP packet

3 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. A method for transformation of network behaviour into parsable input sequence must first be described, which in turn will be quantized using a vector-quantization clustering algorithm to be fed into the LZ78 tree creation algorithm.

3.1 Representation via quantized packet time-difference

The packet time difference is the time elapsed between to packet arrival/departure events in the same flow. Time difference between packets is defined as in (1) below.

$$\Delta_i \triangleq t_{i+1} - t_i \tag{1}$$

Each stream of length k is transformed into a sequence of time differences $\Delta_1, \Delta_2, ..., \Delta_k$ Because $\Delta_i \in \mathbb{R}^+$ is in an infinite range, we wish to reduce the number of time differentials and smooth them over. Thus, vector quantization is performed using K-Means clustering. This enables the use of fewer symbols in the training phase which reduces variance.

3.2 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, the K-Means++ algorithm was used. 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++ [8] 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 the program built, 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, decision boundaries must first be set. For simplicity, they are set halfway between every two centroids.

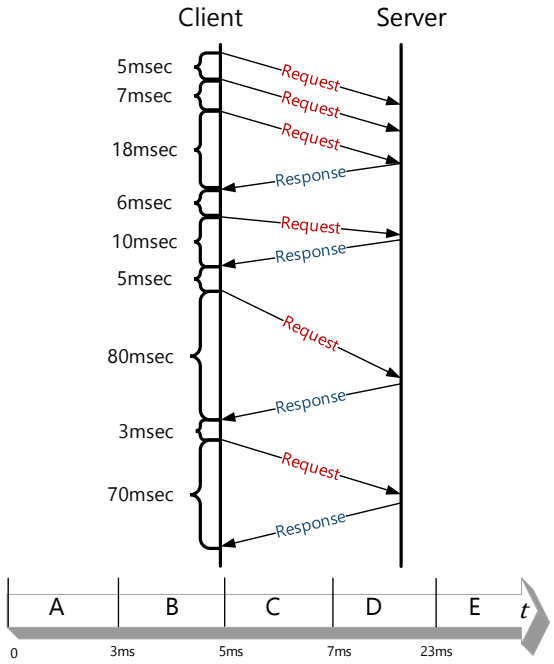


Fig. 4: Packet time difference quantization and letter assignment example

3.3 Classification Methods Examined

3.3.1 Chosen method - Smoothed KL Distance

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

$$D(P||Q) \triangleq \sum_i \ln(P_i/Q_i) \cdot P_i \tag{2}$$

In the proposed algorithm, 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 $Q_i = 0$, causing an undefined value. To counter this case, a modified version called *Smoothed KL Distance* was devised. The algorithm runs on two trees, where T_{FP} is the fingerprint, while T_U is the unknown capture. (The trees are built as described in section 3.4.1)

Algorithm 1 Smoothed KL Distance

procedure SMOOTHEDKL(T_{FP}, T_U)
  $sum \leftarrow 0$
  for all Node n_i in T_{FP} **do**
    $p_i \leftarrow$ probability of node n_i
    if Exact matching node $m \in T_U$ **then**
      $q_i \leftarrow$ probability of node m
    else
      $q_i \leftarrow$ probability of closest node in T_U (lexicographically)
      $q_i \leftarrow q_i \cdot \varepsilon$ # ε is very small
    end if
    $sum \leftarrow$sum+$\log_k(p_i/q_i) \cdot p_i$ # k — amount of centroids
  end for
  return sum
end procedure

3.3.2 Methods not chosen - Hamming Loss and Log Loss

The two techniques Log loss [10] and Hamming loss [11], [12] are functions that represent the cost or value of an event, or an error metric. The error metric is used due to the necessity to predict the time difference for the next packet, based on the context of the time differences seen so far. Since discrete probabilities have already been assigned for each event, these will be the probabilities that will feature in the loss functions.

$$LogLoss = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3}$$

$$HammingLoss(x_i, y_i) = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{xor(x_i, y_i)}{|L|} \tag{4}$$

Where L is the number of labels, D is the number of samples, x_i is the prediction and y_i is the ground truth.

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.

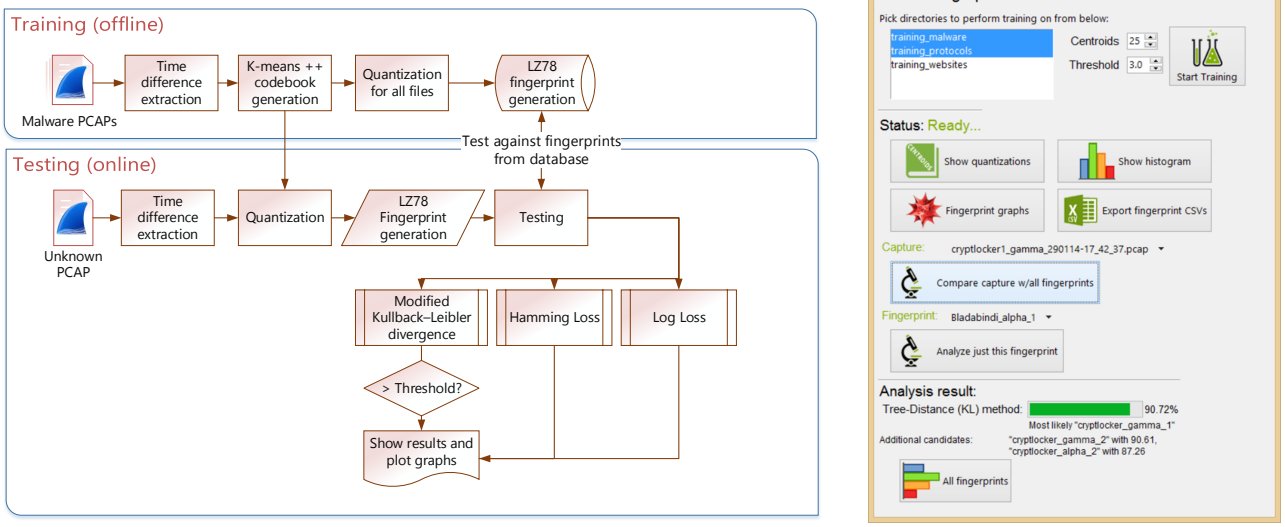
3.4 Program structure and algorithm

A proof-of-concept application was built using Python due to the numerous add-on packages for parsing input files and various mathematical packages. A screenshot of our application is shown in Figure 5b. The program is split into two operating phases: a training phase and a testing phase as seen in Figure 5a.

3.4.1 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[4]. The fingerprints are stored in a local malware fingerprint database for easy access.

4. This action is performed for each malware capture separately



(a) General program structure

(b) Application screenshot

Algorithm 2 Training Algorithm

1: **procedure** TRAINING
2: $GV_O \leftarrow []; GV_C \leftarrow []$
3: $Database \leftarrow []$
4: **for all** Capture$_i$ ←Malware captures **do**
5: APPEND TO($GV_O \leftarrow [\Delta_1, \Delta_2, ..., \Delta_k]$)
6: **end for**
7: $GV_C \leftarrow$K-MEANS++(GV_O)
8: **for all** Capture$_i$ ←Malware captures **do**
9: $V_i \leftarrow \Delta_1, \Delta_2, ..., \Delta_k$
10: Apply quantization transformation $\Delta \Rightarrow c^*$ where $c^* = \arg\min_{c_i} |\Delta - c_i|$
11: Map each quantized value into a corresponding letter from the Latin alphabet (a,b,c,...)
12: $Tree_i \leftarrow$LZ78 TREE GENERATION([c_1,c_2,c_3,...])
13: APPEND TO($Database \leftarrow Tree_i$)
14: **end for**
15: **return** $Database$
16: **end procedure**

3.4.2 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 Algorithm 2 stage 13 — 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.

4 DATASETS AND EXPERIMENTAL SETUP

4.1 Gathering data

4.1.1 Malware dataset

Wireshark captures containing known malware were used. Most were captured by the authors in a sandboxed environment, while additional captures were supplied by a third-party and readily available captures on malware research websites. The captures supplied by third-party sources were identified by the third-party, and contained many varied traffic interspersed: UDP, TCP, HTTP, POP3 and ICMP to name a few.

Captures from malware that communicates to remote machines via WAN were used. A total of 26 malware captures were used to create fingerprints, including Asprox (4 captures), Bladabindi (1 capture), Cryptolocker (10 captures), DarkKomet (3 captures), Expiro (3 captures), Sirefef (1 capture) and Winwebsec (4 captures)

Some of the malware tested attempted to obscure itself as standard HTTP traffic (as described with Cryptolocker in section 2.2), while some access other TCP ports.

4.2 Filtering the captures

Filtering irrelevant entries in the traffic captures required onerous non-automated work.

First, all traffic inside the LAN was removed. WAN IPs were cross-referecned with malware research websites in order to discover if any IPs have already been 'incriminated'. Similar actions were performed on DNS requests to reveal malicious URLs.

All traffic to non-incriminated URL and IPs was then removed, leaving behind the core behaviour of the malware.

4.3 Experimental results

The test setup included training on the malware dataset and control dataset. The number of centroids was set to 25 for all tests. Empirically, we found this value to preform the best. A total of 33 raw captures of malware and other background traffic was collected, being captured on different occasions and different machines. These captures will be tested and verified against the fingerprint database created during training.

The same filtering technique was applied to the raw captures to remove internal LAN communication. The results shown only includes the *Smoothed KL distance* method. The other methods examined did not yield sufficiently certain results. Table A specifies results for comparing an unknown capture versus the fingerprint database using the *Smoothed KL Distance method*. A lower result indicates that the distance between the unknown capture and the fingerprint is smaller, meaning a better match.

Some examples are shown in the ROC graph and Table A below.

4.3.1 The ROC Graph

The ROC graph dispayed in figure 6 visualizes the (for the most part) good performance of the classifier. From the graph, it is shown that for three of the tests (Asprox 2, Asprox 7 and Expiro γ), the True Positive Rate is high for a low False Positive rate.

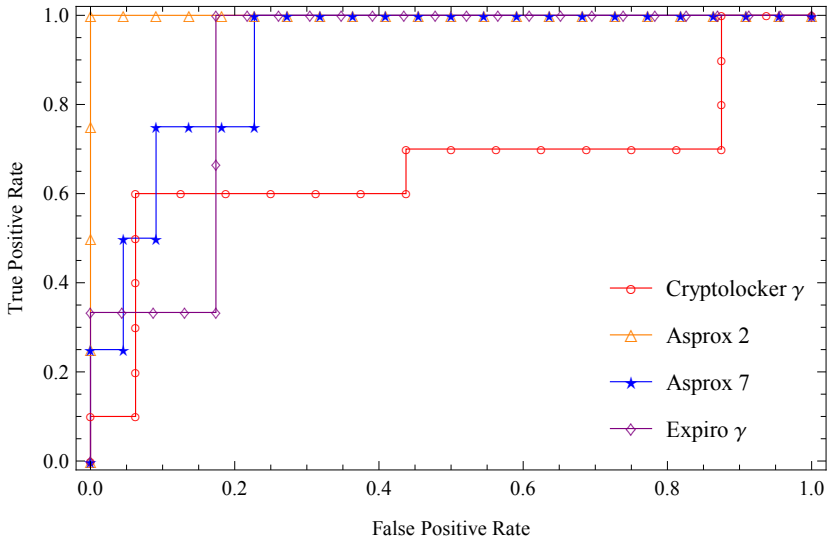


Fig. 6: Characteristic ROC for various captures

4.3.2 Table A

Examining the *Cryptolocker* γ test (the 'worst' performer), while the best match is against a different version of Cryptolocker, there are also good matches against Asprox and Expiro. (See Table A below). For the other fingerprints, the distance value is much larger (> 3.0).

Other malware captures shown below display somewhat better results, with identification occuring with a smaller false-positive rate.

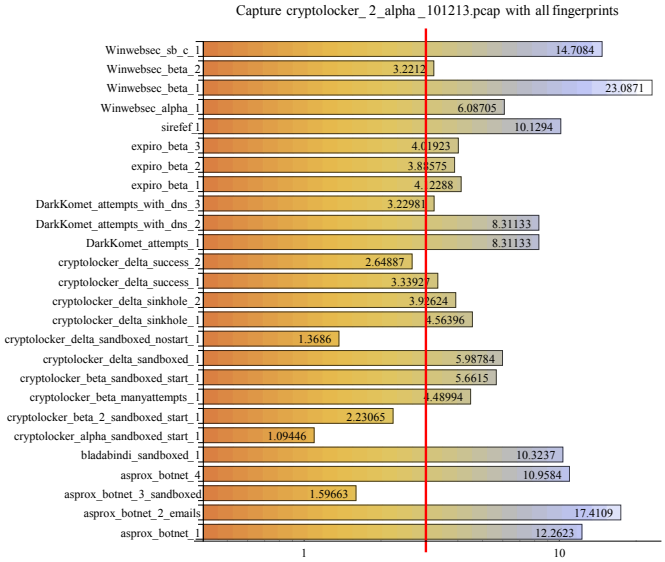
With the *Google* test (clean capture with no known malware traffic), all values are above 3.0, showing that the extracted malware fingerprints are far enough from standard web traffic to avoid a good match result.

Empirically, it was established that results below 3.0 represent a good match. In future works, this parameter can be used as a classifier rule for deciding if a capture contains malware traffic.

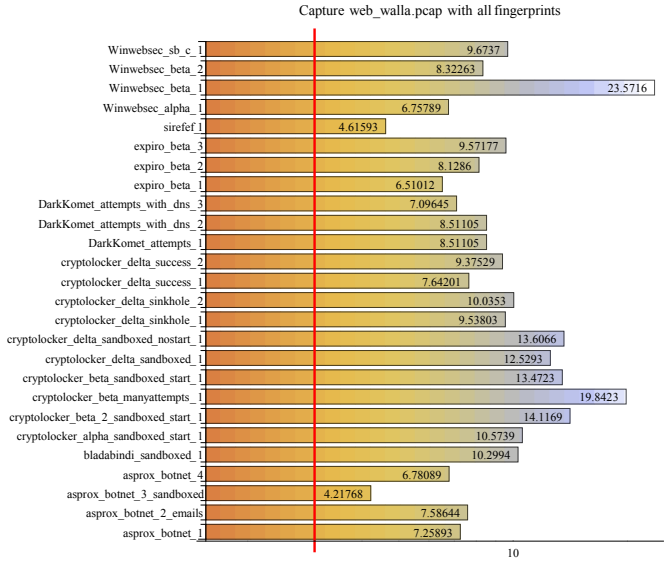
4.3.3 Additional tests

Below are two additional bar-graphs. The bar graphs present a visualization of tests performed on two specific captures - one containing malware traffic and the other malware free. The numbers inside each bar represent the *Smoothed KL Distance* measure, comparing the unknown raw capture with each of the malware fingerprints. A smaller measure represents a better match against the specific fingerprint. As with the numbers presented in Table A, values smaller than 3.0 are considered a good match.

Fig. 7: Bar graphs showing numerical results for Cryptolocker and HTTP connection to Walla.co.il



(a) Graph showing Cryptolocker correctly identified as Cryptolocker malware (smallest value of 1.094).
However, there were also good matches for the Asprox botnet.



(b) Graph showing access to Walla.co.il not being identified as malware.
All values are over 3.0 indicating no probable match.

- The horizontal axes in figures 7a and 7b are logarithmic in order to prevent graphs from being too wide.
- Values smaller than 3.0 (indicated by red vertical line) considered to be good matches. The smaller the number, the better the match.

5 Conclusion

We proposed a program for identifying malware traffic using only time-differences seen in computer network traffic, hopefully eliminating the need for DPI. In order to build a probability model, the LZ78 tree generation algorithm was used. The decision making process is based on a proposed modification of the KL Distance algorithm. These techniques were applied to timing differences seen in computer network behaviour.

Using these techniques has shown that given a timing sequence identified as malware, the proposed application has been able to successfuly identify malware behaviours.

The system was tested using real-world traffic captures containing interspersed malware traffic and real-world captures containing no malware traffic.

Because of the laborious fingerprint and capture process, the number of tests performed was limited. While the comparison method offered shows good results in the scenario tested, further testing in more elaborate scenarios is needed in order to further identify the shortcomings of this technique.

In future works, it is possible to enhance the fingerprint by using more data such as packet header-sizes, session lengths and sizes and more in order to improve detection rates. In our work, the Hamming-loss and Log-loss algorithms did not function in a consistent manner. Further experimentation and alterations on these methods should be considered in future works, as they may yet produce better results in some scenarios.

APPENDIX

TABLE A: TEST RESULTS FOR VARIOUS RAW CAPTURES

	Cryptolocker γ (29/01/2014)	**Asprox 2**	**Asprox 7**	**Google** website	**Expiro** γ (31/07/2014)
asprox botnet 1	7.59902	**2.78475**	5.55845	7.18549	6.49411
asprox botnet 2 emails	14.1443	**0.464713**	7.07292	6.70332	10.0197
asprox botnet 3 sandboxed	**0.798521**	**0.465128**	**2.68669**	**4.15564**	3.57509
asprox botnet 4	**2.68481**	**2.14358**	**0.453436**	4.66969	3.51468
bladabindi sandboxed 1	6.63192	6.76329	6.43112	9.02466	5.83629
cryptolocker α sandboxed start 1	7.79634	7.2265	8.8693	10.6224	8.04624
cryptolocker β 2 sandboxed start 1	10.9085	10.3631	12.5386	13.73	9.65315
cryptolocker β manyattempts 1	4.1826	14.2357	18.8726	19.347	9.42026
cryptolocker β sandboxed start 1	**1.37924**	9.86885	12.4388	14.2139	9.21342
cryptolocker δ sandboxed 1	**0.480212**	11.4516	12.9203	14.5532	3.45415
cryptolocker δ sandboxed nostart 1	11.0304	10.7269	12.7308	13.6632	9.97803
cryptolocker δ sinkhole 1	**1.45018**	9.32625	9.82628	12.6773	4.51411
cryptolocker δ sinkhole 2	**1.07158**	5.12814	7.42928	9.93804	2.52219
cryptolocker δ success 1	**1.87088**	6.58319	9.00866	9.4286	**1.25508**
cryptolocker δ success 2	0.883449	8.53382	8.12474	11.7019	**1.00032**
DarkKomet attempts 1	6.43907	8.06885	7.68363	8.54571	10.7481
DarkKomet attempts with dns 2	6.43907	8.06885	7.68363	8.54571	10.7481
DarkKomet attempts with dns 3	4.70849	4.03795	5.83051	7.11032	5.22355
expiro β 1	**2.09184**	5.35655	7.89889	10.9454	**0.602286**
expiro β 2	**2.86033**	**2.92171**	3.8821	6.57786	3.01845
expiro β 3	3.87015	5.67466	8.79663	9.94486	**2.71112**
sirefef 1	3.71103	4.00045	**2.15289**	5.40148	5.64391
Winwebsec α 1	5.64786	3.16778	7.58029	6.77432	3.2514
Winwebsec β 1	13.667	13.5089	16.3944	16.7452	7.3596
Winwebsec β 2	3.16557	4.95311	6.42347	7.3632	**1.68997**
Winwebsec sb c 1	9.30291	9.71032	9.47729	11.4866	8.20818

Notes:

- **Bold-underline figures** mark best match.
- **Bold figures** mark good matches
- All raw captures differ from captures used for training.

ACKNOWLEDGMENTS

The authors would like to thank CYREN for granting remote access to their lab and malware samples.

REFERENCES

[1] Avraham Lempel and Jacob Ziv. Compression of individual sequences via variable-rate coding. *IEEE Transmissions on Information Theory*, page 530536, September 1978.

[2] Meir Feder and Neri Merhav and Michael Gutman. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, 38(4):12581270, July 1992.

[3] Mordechai Nisenson and Ido Yariv and Ran El-Yaniv and Ron Meir. Towards Behaviometric Security Systems: Learning to Identify a Typist. *Knowledge Discovery in Databases: PKDD 2003, Lecture Notes in Computer Science*, 2838:363374, 2003.

[4] Jr Langdon G.G. A note on the Ziv - Lempel model for compressing individual sequences (Corresp.). *IEEE Transactions on Information Theory*, 29(2):284287, 1983.

[5] Asaf Cohen and Shlomi Dolev and Niv Gilboa and Guy Leshem. Anomaly Detection, Dependence Analysis and. Technical Report, Ben Gurion University of the Negev, Beersheba,, 2012.

[6] Nart Villeneuve and James Bennett. Detecting APT Activity with Network Traffic Analysis. 2012.

[7] EMSISoft Blog. CryptoLocker - a new ransomware variant Available: *http://blog.emsisoft.com/2013/09/10/cryptolocker-a-new-ransomware-variant/*, [Accessed 16 August 2014]

[8] David Arthur and Sergei Vassilvitskii. k-means+: The advantages of careful seeding. *Proceedings of the eighteens annual ACM-SIAM symposium on Discrete algorithms*, page 10271035, 2007.

[9] Solomon Kullback and Richard Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):7986, 1951.

[10] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, Cambridge, UK, 2006.

[11] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview *International Journal of Data Warehousing and Mining*, 3(3):1-13, 2007.

[12] Richard W Hamming. Error detecting and error correcting codes. *Bell Systems technical journal*, 29(2):147160, 1950.