J Comput Virol (2011) 7:279–295
DOI 10.1007/s11416-011-0156-6

ORIGINAL PAPER

Classification of packet contents for malware detection

Irfan Ahmed · Kyung-suk Lhee

Received: 15 November 2010 / Accepted: 5 October 2011 / Published online: 22 October 2011

Abstract Many existing schemes for malware detection are signature-based. Although they can effectively detect known malwares, they cannot detect variants of known malwares or new ones. Most network servers do not expect executable code in their in-bound network traffic, such as on-line shopping malls, *Picasa, Youtube, Blogger*, etc. Therefore, such network applications can be protected from malware infection by monitoring their ports to see if incoming packets contain any executable contents. This paper proposes a content-classification scheme that identifies executable content in incoming packets. The proposed scheme analyzes the packet payload in two steps. It first analyzes the packet payload to see if it contains multimedia-type data (such as `avi, wmv, jpg`). If not, then it classifies the payload either as text-type (such as `txt, jsp, asp`) or executable. Although in our experiments the proposed scheme shows a low rate of false negatives and positives (4.69% and 2.53%, respectively), the presence of inaccuracies still requires further inspection to efficiently detect the occurrence of malware. In this paper, we also propose simple statistical and combinatorial analysis to deal with false positives and negatives.

1 Introduction

Malware (such as worm, virus, rootkit) is a malicious executable code that infiltrates computer system without authorization. Many existing schemes for malware detection are signature-based [1,3,20,28]. Although they can effectively detect known malwares, they cannot detect variants of known malwares or new ones. In order to detect an unknown malware it must first be analyzed to derive a signature. When the signature is finally made public, the malware would have already infiltrated many hosts. Therefore, to detect 0-day exploits effectively and efficiently, we need a different approach that looks for common characteristics among malwares regardless of whether they are known or not. One such characteristic is that many malwares are executable codes.

This motivates us to distinguish between executable and non-executable content in general. Such approach is effective to protect many network applications (from binary malwares) that do not expect to receive any executable code. For instance, malwares particularly target network servers for mass-spreading because they stay on-line and are open to unknown users. However, most servers are data providers in nature and do not expect executable code in their in-bound network traffic. Some of their examples are as under:

- Web servers such as on-line shopping malls only accept http requests, which are simply text.
- Many applications that accept user uploads do not expect to receive executable code. For instance, *Picasa* (a photo sharing website) only allows sharing of image files. *Youtube* (a video sharing website) only allows sharing and uploading of videos. In *Blogger* (a blog website), users maintain their commentaries or descriptions of events and share images and videos.
- There are grid applications and distributed computing environments which exchange a few well-defined types of data such as multimedia or numeric, but do not exchange executable binaries.

I. Ahmed
Information Security Institute, Queensland University of Technology, Brisbane, Australia
e-mail: irfan.ahmed@qut.edu.au

K. Lhee (✉)
Seoul, South-Korea
e-mail: kyungsuk.lhee@gmail.com

– Many FTP servers that allow users to download multimedia contents such as movies are not supposed to transfer executable content. Therefore, such network applications can be protected from malware infection by monitoring their ports to see if incoming packets contain any executable contents.

Many server machines are sandboxed and/or simplified to contain a minimal set of services and programs. For example, if compilers or interpreters (such as Java, Perl, etc.) are unavailable in the server machine, then the attacker has no choice but to rely on binary malware. Therefore, the approach of identifying executable content in packet payload can be applicable to many network applications.

A universal solution for all kinds of malware is extremely difficult to realize and thus may not be feasible in near future. Rather, a few solutions that are (ideally) collectively exhaustive in the attack space would be more achievable. Therefore the proposed approach is intended to complement other approaches such as signature-based approach.

This paper proposes a content-classification scheme that identifies executable contents in incoming packets. The proposed scheme has the aforementioned advantages over signature-based approach. It is also easier to deploy than similar existing solutions, since most of them require port-specific learning phases [10,13,24,25]. In their approaches, statistical parameters of packet contents on a particular port may change over time and therefore also require ongoing analysis. The proposed scheme relies on statistical parameters of executable files rather than port-specific information and therefore can be readily used in any port, as long as the protected network application and the port it is bound to do not expect executable content.

For accurate detection, the proposed scheme analyzes the packet payload in two steps. It first analyzes the packet payload to see if it contains multimedia-type data (i.e., binary contents except executables such as `avi`, `wmv`, `jpg`, etc.). If not, then in the second step it classifies the payload either as text-type (`txt`, `jsp`, `asp`, etc.) or executable-type data (see Fig. 1). We propose a two-step scheme because we found that the statistical parameters of multimedia-type are different enough from text and executable-type, but the difference between text and executable-type is smaller, and therefore different techniques are required to distinguish them.

The two algorithms in the proposed scheme are based on the *n*-gram, which is a sequence of *n* adjacent bytes in a packet payload. To identify multimedia content, we use a distance-based algorithm. We first obtain a threshold value (that represents the multimedia-type) from the *n*-gram frequencies of numerous executable contents (which are the *model* of the executable-type). From each packet we then calculate a *distance* value from the *n*-gram frequencies of the packet payload and the model (which shows how different the packet is from the model), and compare it with the threshold. If the distance value exceeds the threshold, it is considered as multimedia-type. Otherwise we apply the second algorithm to identify whether or not it contains executable content. To identify executable content, we propose our *pattern counting* algorithm. We first obtain a set *S* of unique *n*-gram patterns that are present only in executable contents. From each packet payload we then count the occurrence of *n*-gram patterns in *S*. If the count exceeds a certain threshold then it is considered as executable-type, otherwise as text-type. Our pattern counting algorithm is particularly effective for distinguishing text-type, because text contents have considerably fewer *n*-gram patterns than other content types.

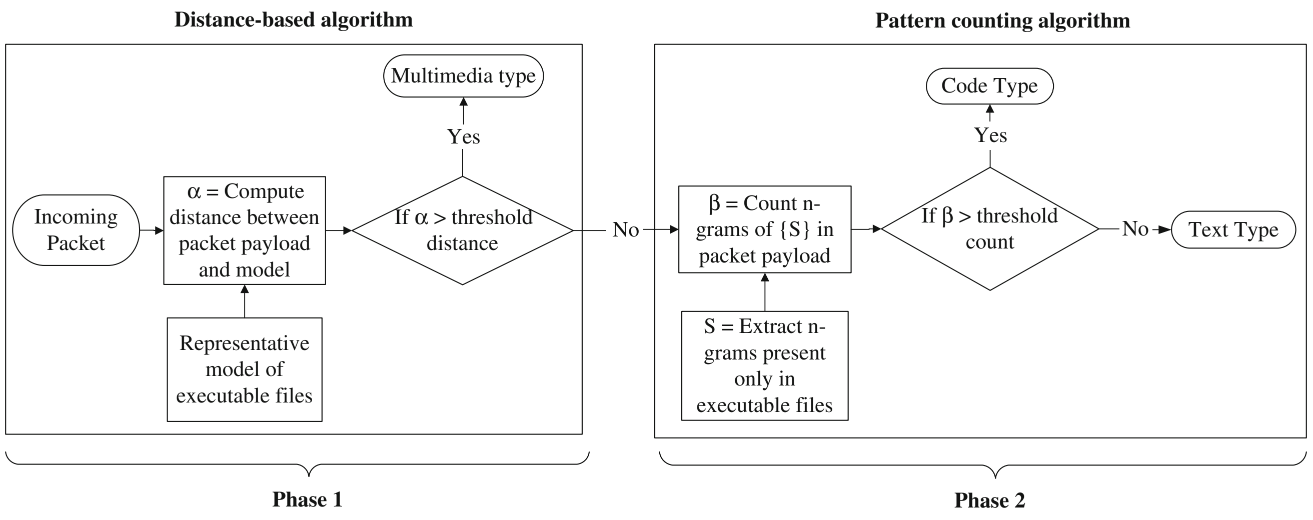

Distance-based algorithm
Pattern counting algorithm
Multimedia type
Yes
Incoming Packet
α = Compute distance between packet payload and model
If α > threshold distance
No
Representative model of executable files
Code Type
Yes
β = Count n-grams of {S} in packet payload
If β > threshold count
No
Text Type
S = Extract n-grams present only in executable files
Phase 1
Phase 2

Fig. 1 Outline of the proposed content classification scheme. *Phase 1* detects multimedia-type contents, and *Phase 2* identifies executable contents

We conducted experiments simulating real-time situations in order to measure the false positive and negative ratio, using a broad range of contents such as documents, videos, images, executables and binary malwares. Although the proposed scheme shows a very low rate of false negatives and positives (4.69% and 2.53%, respectively), the presence of inaccuracies requires further inspection to efficiently detect the occurrence of malware. In this paper, we also propose simple statistical and combinatorial analysis to deal with false positives and negatives.

Contributions and important points

- The proposed solution specifically focuses on distinguishing the (malignant or benign) executable code from non-executable contents. This is highly applicable to the in-bound network traffic of the network applications that do not expect to receive any executable code and thus, enables them to detect binary malwares in their in-bound traffic.
- The part of the proposed solution that uses Mahalanobis distance to identify multimedia contents is similar to the Wang et al. [25] and Lee et al.s' [10] solutions. However, they use it for anomaly and botnet detection respectively. They also use port-specific network packets to build models, on contrary to ours where we use arbitrary executable files.
- The pattern counting scheme is novel to exploit the significant difference in the number of n-grams between text and executable contents.
- We conduct the experiments with unbiased data set that contains the files of diverse as well as similar types. For instance, we use text file types such as `txt`, `xml` and `html`; Microsoft office compound files such as `xls` and `doc` that contain embedded objects and multimedia files such as `wmv`, `mp3`, and `rm`. All the groups are quite different in contents from each other.
- The solution is currently effective for distinguishing the unaltered contents. The contents after encoding, encryption or zipped for instance, are not in the scope of this paper (and will be considered in the future work).

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the distance-based algorithm to identify multimedia packets. Section 4 describes the pattern counting scheme to identify executable and text packets. Sections 5 and 6 show the experiment results. Section 7 presents simple statistical and combinatorial analysis to deal with false positives and negatives. Section 8 discusses memory overhead, and Sect. 9 concludes the paper and discusses our future work.

2 Related work

In this section we discuss existing content classification schemes for malware detection and file type identification. To maintain the focus, we only discuss the work that classifies the contents using statistical or machine learning techniques.

2.1 Anomaly and malware detection

Wang and Stolfo proposed PAYL [25], which examines the packet payload and computes its *1*-gram frequency distribution. It builds many models of normal network traffic, depending on the packet size and monitoring port, by calculating the average and standard deviation of the feature vector of normal traffic. It uses the Mahalanobis distance to compare the normal traffic model with a *1*-gram (byte) frequency distribution of test packets, to detect worms in packet payloads. They used *1999-DARPA-IDS* dataset and (at best) achieved 100% accuracy with 0.1% false positive rate for port 80. They also proposed Anagram [24] which works in a similar manner but uses a higher n-gram (using the bloom filter) to detect mimicry attacks. In some cases, they reported 100% accuracy with 0.006% false positive rate for Anagram.

Bolzoni et al. [7] extended PAYL [25] (which they named as POSEIDON) to obtain better anomaly detection rate. They proposed two tier architecture where the first tier used Self-Organizing-Maps to classify payload data in an unsupervised manner. The second tier is a modified PAYL that builds the representative models of normal traffic with respect to the classified data (instead of the payload length as used by traditional PAYL), destination address and service port. They compared POSEIDON with traditional PAYL and reported the detection rate of 73.2 and 58.8%, respectively.

Wenke Lee et al. [10] proposed SLADE that is similar to PAYL [25] except that it does not use all features of an n-gram, e.g. a *1*- and *2*-gram has 256 and 2,562 features, respectively. It uses a lossy structure, i.e. a fixed vector counter to store a lossy n-gram distribution of payloads. They use a lossy structure because they conjecture that not all the n-grams represent normal traffic behavior. Some n-grams are considered as noise. They compare SLADE with PAYL and reported 0.3601 and 4.02% false positives, respectively, with 100% accuracy.

Zanero [27] proposed `ULISSE` which is a network anomaly detector. It is a two-tier architecture where the first-tier used Self-Organizing-Maps with Euclidean distance to group the packets in such a way that intra-group similarity is maximized and inter-group similarity is minimized. The second tier used a multivariate time series outlier detector for anomaly detection. They reported the best detection rate of 88.9% while protecting an Apache web server against the attacks that he generated from `metasploit`. He used source port, destination port, TCP flags, source and destination address and the classified payload (from the first tier) as features to the outlier detector.

Criscione et al. [9] proposed Masibty, an anomaly-based intrusion prevention system for web applications. The system is based on *Entry points* which they define as an augmented URI (identifier of the requested resource from web server) by parameters and session context. Masibty has several *Anomaly Engines* (each of which detects anomaly of a single aspect of an event). Anomaly Reasoner obtains anomaly scores from Anomaly Engines, aggregates them and compares the aggregated score from user-specific threshold in order to flag the events as anomalies. They also have other modules (such as Normality Vault and Reaction Manager) to improve the performance. They used Artmedic Weblog, SineCMS, PHP-Nuke and JAF web applications to test Masibty and reported overall detection rate of 93% and false positive rate of 0.16%.

Stolfo et al. [21] use their previous work of file type identification [15] to detect malwares that are only embedded in pdf and doc files. They also classify normal executables and viruses in a similar fashion. They use 1- and 2-gram frequency distributions and build the model by using 1,000, 500 and 200 bytes that are truncated from the header and trailer of benign executable and virus files. They reported the detection accuracy of 87.5, 90.5, and 94.5% for truncated bytes from header and 75, 80.1, and 72.1% for truncated bytes from trailer, respectively.

Li et al. [14] proposed two techniques to detect malwares embedded in Microsoft Word document files (.doc), i.e. static content analysis and run-time dynamic testing. In static content analysis, they model sample documents and malwares by storing all their *n*-grams in two different models. They used 5-gram that is found to be detecting attacks accurately while consuming a reasonable amount of memory. They reported 7.68% false negatives and 0.02% false positives. They further extend their approach by parsing the training document, dividing it into different sections and generating one model for each section. They reported 1.31% false negatives and and 0.15% false positives for this approach.

Shafiq et al. [19] observe that, unlike malwares, the byte sequences in benign files have first-order dependence. That is, when byte *i* appears, it is more likely that it will be followed by another *i* at the next byte location. They use a discrete-time Markov chain that characterizes a process in terms of the conditional distribution of its state. The transition probabilities are computed by counting the number of times byte *i* is followed by byte *j* for all *i* and *j*. They use the entropy rate to quantify the changes in the transition probability matrix. The entropy highlights the perturbations in the location where malware is embedded in a file. They used DOC, EXE, JPG MP3, PDF and ZIP file types in their experiments and created two data sets: first by embedding malwares in random locations of files and second by encrypting malwares prior embedding them in random locations of files. They reported the improvement from Stolfo et al. [21] in detection of the non-encrypted and encrypted malwares (in terms of true positives) by 0.7 and 10.1% for DOC, 30.8 and 28.4% for EXE, 19.1 and 25.7% for JPG, 31.2 and 37.5% for MP3, 9.1 and 21.6% for PDF, and 30.4 and 35.1% for ZIP, respectively. However, their scheme was ineffective for compound file types such as DOC. Moreover, they also reported that their scheme couldn't detect malwares smaller than 343 bytes when a block size of 1,000 bytes was used for tracing the malware in a benign file.

SigFree [26] proposed two schemes that distinguish between a sequence of random instructions and a fragment of a program in machine language. Scheme 1 uses the operating system characteristics such as calls to operating systems and kernel library. Scheme 2 on the other hand exploits the data flow characteristics of a program. They disassemble and extract the instruction sequences in packet payloads up to a certain threshold, in order to find executable codes. They find the appropriate thresholds by testing both the schemes against 50 encrypted attack requests generated by metasploit framework, worm Slammer, CodeRed and 1500 binary HTTP replies (containing encrypted data, audio, JPEG, GIF, and PNG). They reported that by setting a threshold number of push-calls to two, scheme 1 detected all the buffer flow attacks (used in the experiments). Moreover, by setting the threshold of the sequence of instructions for scheme 2 between 15 and 17 detected all the attacks (in the experiments).

Kruegel et al. [13] proposed an anomaly based intrusion detection system. They built application-specific models by using only application data in packet payload. Rather than using a byte frequency distribution of all byte patterns, they used six ranges (0, 1–3, 4–6, 7–11, 12–15, and 16–255) of byte frequency distributions in packet payloads. They computed a single distribution model of these six segments and used a Chi-square test on this model to detect anomalies. Their tests included five DNS exploits and showed that the distribution of malicious DNS requests had anomaly scores quite different (or greater) than normal DNS requests and thus, by setting an appropriate threshold, all the malicious requests were accurately detected.

Table 1 presents the comparison of our solution with the other packet-payload based intrusion detection systems in order to highlight the significant differences between them.

2.2 File type identification

McDaniel and Heydari [18] introduced three algorithms to analyze file content and identify file types. Firstly, the Byte-Frequency Analysis Algorithm (BFA) computes the byte frequency distributions of different files and generates a "fingerprint" of each file type by averaging the byte-frequency

Table 1 Comparison of existing packet-payload based approaches and our proposed approach

	Online learning	Packet header	Learning type	Learning source	Attack-free traffic	Port information for learning	Detecting attacks
PAYL	Required	Required	Unsupervised	Packets	Required	Required	Payload-based attacks
POSEIDON	Required	Required	Unsupervised	Packets	Required	Required	Payload-based attacks
ULISSE	Required	Required	Unsupervised	Packets	Required	Required	Web server attacks
Masibty	Required	Not-required	Unsupervised	–	Not-Required	Not-required	Web application attacks
Our approach	Not-required	Not-required	Supervised	Files	Not-required	Not-required	Executable contents (and/or binary malwares)

distribution of their respective files. They also calculate correlation strength as another characterizing factor by taking the difference of the same byte in different files. Secondly, the byte-frequency cross-correlation (BFC) algorithm finds the correlation between all byte pairs. It calculates the average frequencies of all byte pairs and the correlation strength in a similar manner to the BFA algorithm. Thirdly, the file header/trailer (FHT) algorithm uses the byte-patterns of file headers and trailers that appear in a fixed location at the beginning and end of a file, respectively. In these algorithms, they compare the file with all the generated fingerprints to identify its file type. They reported the accuracy of 27.50, 45.83, and 95.83% for BFA, BFC and FHT algorithms, respectively.

Karresand and Shahmehri [16,17] proposed the "Oscar" method to identify a file fragment type. They use the single centroid model [15] of Li et al. and use quadratic distance metric and 1-norm as a distance metric to compare the centroid with the byte-frequency distribution of a given file. Although their method identifies any file type but they have specifically optimized it for `jpg` files. They reported a 99.2% detection rate with no false positives.

Veenman [23] uses linear discriminant analysis to identify file types. Three features are obtained from file content, i.e. the byte frequency distribution, the entropy derived from the byte-frequency distribution of files, and the algorithmic or Kolmogorov complexity that exploits the substring order [12]. Calhoun and Coles [8] extended Veenman's work by building classification models based on the ASCII frequency, entropy and other statistics, and applied linear discriminant analysis to identify file types. Veenman reported 45% overall accuracy.

Ryan [11] uses a neural network to identify file types. He divides the files into blocks of 512 bytes, and uses only the first 10 blocks for file-type identification. Two features are obtained from each block, i.e. raw filtering and character code frequency. Raw filtering takes each byte as an input to one neuron of the neural network. On the other hand, character code frequency counts how many times each character code occurs in the block and takes the frequency of characters as input to the neurons. He used only image files (such as `jpg`, `png`, `tiff`, `gif`, and `bmp`) as a sample set and reported a detection rate ranging from 1% (`gif`) to 50% (`tiff`) when using raw filtering and from 0% (`gif`) to 60% (`tiff`) when using character code frequency.

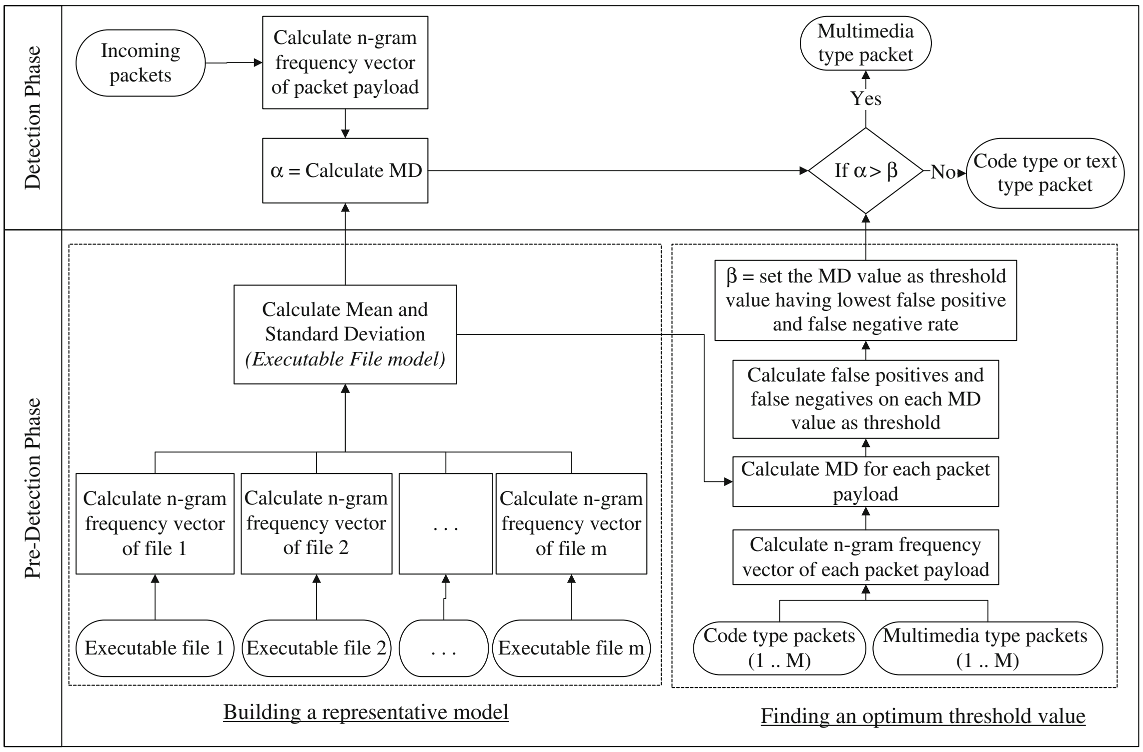
Mehdi et al. [6] use the hierarchical feature-extraction method to better exploit the byte-frequency distribution of files in file-type identification. They utilize principal component analysis and an auto-associative neural network to reduce the 256 features of byte patterns to a certain small number for which the detection error is negligible. After feature extraction, they use the three layers MLP (multi-layer perceptron) for detecting the file types. They used `doc`, `pdf`, `exe`, `jpg`, `html` and `gif` file types for experiments and reported an accuracy of 98.33%.

3 Identification of multimedia packets

This section describes the distance-based algorithm to identify multimedia content in the packet payload, which requires a learning phase to build a normal behavior model and an identification phase. We also discuss the order of n-gram that yields sufficient accuracy in identifying multimedia contents (we found that a 3-gram is sufficient). Figure 2 illustrates the algorithm.

3.1 Learning phase

In the learning phase, we build an n-gram model of the executable contents based on the observation that the executable contents have similar byte patterns (as shown in Fig. 3). To build the model, we first obtain the n-gram frequency vectors from numerous executable files. Each file is processed to compute the number of occurrences (frequency) of each n-gram, which is then normalized by dividing the frequency by the file size (relative frequency). Since it is hard to find the real instructions from executable file (as they are often confused with the normal data [26]), we consider the whole file to obtain the n-gram frequencies of executable files. However, we conjecture that only considering n-grams corresponding to real instructions could obtain a better representative



Detection Phase
Incoming packets
Calculate n-gram frequency vector of packet payload
α = Calculate MD
Multimedia type packet
Yes
If α > β
No
Code type or text type packet
Pre-Detection Phase
Calculate Mean and Standard Deviation (Executable File model)
Calculate n-gram frequency vector of file 1
Calculate n-gram frequency vector of file 2
. . .
Calculate n-gram frequency vector of file m
Executable file 1
Executable file 2
. . .
Executable file m
β = set the MD value as threshold value having lowest false positive and false negative rate
Calculate false positives and false negatives on each MD value as threshold
Calculate MD for each packet payload
Calculate n-gram frequency vector of each packet payload
Code type packets (1 .. M)
Multimedia type packets (1 .. M)
Building a representative model
Finding an optimum threshold value

Fig. 2 Distance-based algorithm to identify multimedia contents

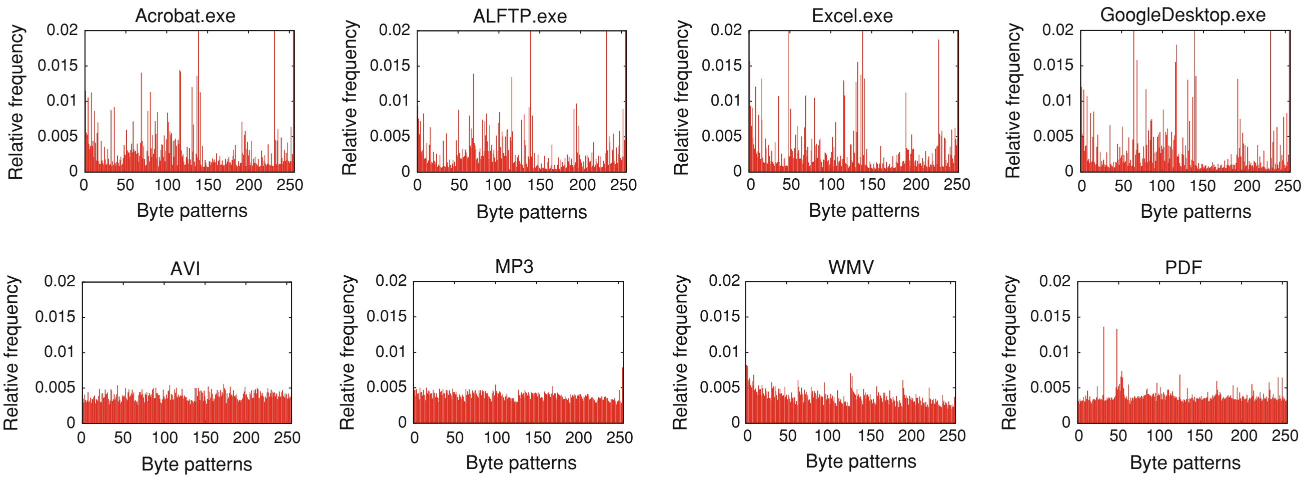


Fig. 3 *1*-gram (byte) frequency vector of executable and multimedia-type (i.e. `avi`, `mp3`, `wmv`, `pdf`) files

frequency distribution of executable files. (Therefore, the accuracy level measured in this paper serves as a lower bound.) After normalization, considering each *n*-gram's relative frequency as a variable, we compute its mean and standard deviation. We refer to the set of mean and standard deviation of all *n*-grams as the *executable model*.

The executable model is compared with the *n*-gram frequency vector of incoming packets by using the Mahalanobis distance. The Mahalanobis distance between two vectors $\overrightarrow{x}$ and $\overrightarrow{y}$ is defined as

$$d(\overrightarrow{x},\overrightarrow{y}) = \sqrt{(\overrightarrow{x}-\overrightarrow{y})P^{-1}(\overrightarrow{x}-\overrightarrow{y})^T} \qquad (1)$$

where P^{-1} is the inverse of the covariance matrix of the data [22]. The Mahalanobis distance is widely used for detecting outliers, thus it is useful for anomaly detection. However computing the Mahalanobis distance is expensive, so we use the simplified Mahalanobis distance (denoted as

MD henceforth) formula in PAYL [25] which considers both the average and standard deviation of the variables measured. The MD equation is as follows.

$$d(x, \bar{y}) = \sum_{i=1}^{N} \frac{|x_i - \bar{y}_i|}{\bar{\delta}_i} \tag{2}$$

where $\bar{y}$ is the mean value, $\bar{\delta}$ is the standard deviation (from the executable model), x is the n-gram frequency of the incoming packet and N is the total number of unique patterns of the n-gram.

Since the byte-frequency distributions of executable contents are quite distinct from multimedia contents (see Fig. 3), we can obtain a significant difference between the Mahalanobis distance of executable and multimedia contents by making a comparison with the executable model. Therefore, the learning phase also involves setting an optimum threshold value between the Mahalanobis distance values of two content types. To compute the threshold value, we compute the n-gram frequency vector of a packet payload of known type (code and multimedia) and use the Mahalanobis distance to compare it with the executable model. After calculating the MD of the code and multimedia-type packets, we compute the false negative and false positive rate for all possible MD thresholds, and set the optimum threshold that yields the minimum rate of false negatives and false positives.

3.2 Identification phase

During the identification phase, for each incoming packet we compute the n-gram frequency vector of the payload by dividing the count of each n-gram by the packet size. This vector is then compared with the executable model by using the MD, which shows how consistent the packet payload is with the executable model. If the MD value is higher than the predefined threshold, the packet is considered as a multimedia-type. If the MD value is lower, the packet may be either code type or text type. That is, there is still ambiguity between the text and code type, which requires further processing to resolve.

3.3 Determining the n-gram order

An instruction consists of multiple bytes (even an opcode alone may consist of multiple bytes), so byte sequence information is significant in identifying the codes in a packet. Therefore, a higher order n-gram is desirable. We increase the order of the n-gram until we can reasonably distinguish the code and other types of packets. Therefore, for each order of n-gram, we process different types of packets, including packets containing executable code, to calculate the threshold. We build the executable model using 134 program files with sizes ranging from 330 bytes to 17,471 KB, and compare it with the different types of packets. Figure 4 shows the MD of different types of packet, for *1*-, *2*- and *3*-grams. The MD of code packets must be below the

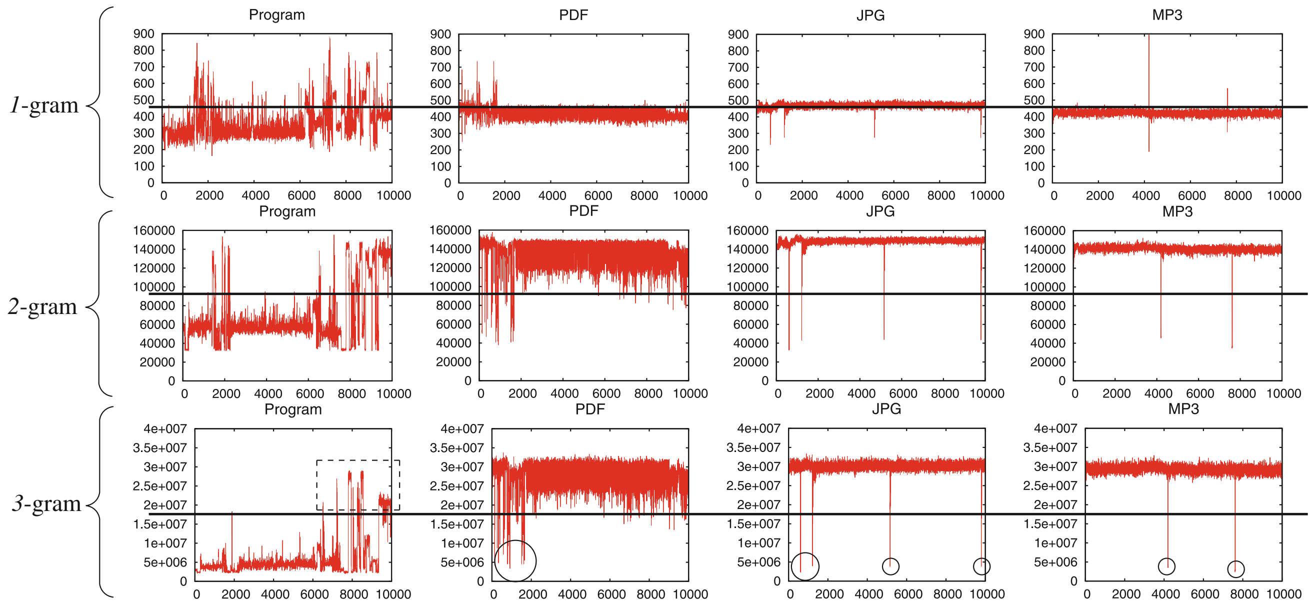


Fig. 4 MD vectors of code and multimedia packets, using *1*-, *2*-, and *3*-grams (the x-axis shows the number of packets processed; the y-axis shows the MD calculated for each packet; the *horizontal lines* in the graph show the thresholds for the best overall accuracy (i.e. the least sum of false positives and negatives)). The *circles* and *dotted box* represent false negatives and positives, respectively

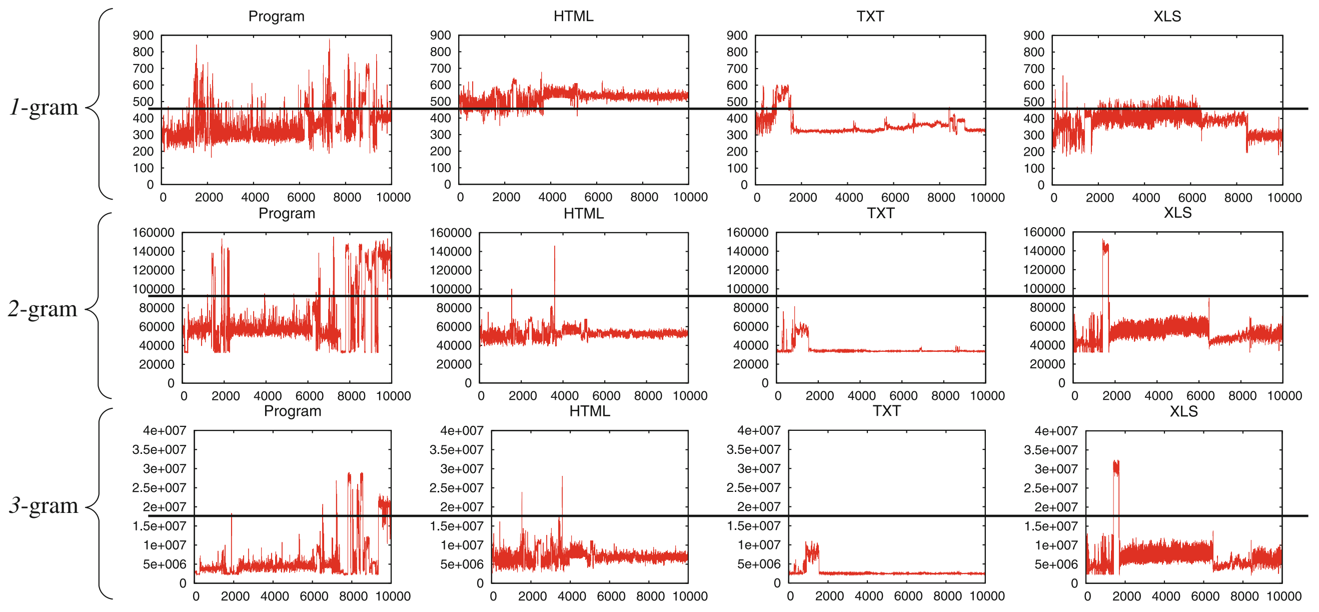


Fig. 5 MD vectors of code and text packets, using *1*-, *2*-, and *3*-grams (the *x*-axis shows the number of packets processed; the *y*-axis shows the MD calculated for each packet; the *horizontal lines* in the graph show the thresholds). Note that it is not possible to draw a meaningful threshold line

threshold of multimedia-type packets. Figure 4 shows that a *3*-gram can better distinguish between code and multimedia packets.

3.4 Limitation of the distance-based algorithm

We found that the distance-based algorithm using the executable model can identify multimedia types, but cannot distinguish text from codes because we obtain similar MD values from both code and text packets. Thus, it is difficult to find a threshold that clearly distinguishes code from text packets. We also tried different orders of *n*-gram, but were unable to obtain the significant difference in MDs, needed to distinguish them (see Fig. 5). Therefore, we conclude that MD is an insufficient quantification measure for differentiating between code and text packets, and instead propose an additional scheme to resolve this ambiguity, which is discussed in the following section.

4 Identification of executable and text packets

In this section, we present two algorithms that can resolve the type of non-multimedia packets. The first is a distance-based algorithm (although it uses text files in the learning phase instead of executable files), and the second is a novel algorithm that we call the *pattern counting scheme*. While the second algorithm has superior performance, we also discuss the first one for the purpose of comparison.

4.1 Distance-based algorithm for text contents

Our first attempt uses the same distance-based algorithm presented for identifying multimedia packets, except that the model is built using text files instead of executables as follows.

1. In the learning phase, we first compute the *n*-gram frequency vectors of text files, from which we then calculate two vectors (mean and standard deviation). This called the *text model*.
2. In the identification phase, from each incoming packet we compute a MD value using the *n*-gram frequency vector of the packet payload and the text model. If the MD is lower than a certain threshold, the packet is considered as text, otherwise as code.

However, as shown in Sect. 5.2, we cannot achieve sufficient classification accuracy with this scheme.

4.2 Pattern counting scheme

This scheme is based on the observation that there are a number of *n*-gram patterns that normally occur in executable contents but rarely occur in text contents. It is because unlike executable contents, text-type contents use limited range of byte patterns which are ASCII or printable characters (such as [1–9], [a–z] and [A–Z]). This exponentially amplifies the difference of normally-occurred *n*-grams between executable and text contents.

Table 2 Number of unique *3*-grams in various types of files

Files classification	File type	Number of *3*-gram permutations in files (A)	B (%) = (A/total number of *3*-gram permutations 100)
Text files	xml	13,926	0.08
	html	33,845	0.20
	asp	47,270	0.28
	jsp	47,867	0.28
	php	53,469	0.32
	txt	54, 427	0.32
	xls	692,463	4.13
	doc	6,696,694	39.92
	AVG	954,995.1	5.69
Executable file	Program	16,654,060	99.26

AVG is the average of each file classification

This scheme computes the frequency of such patterns in a packet payload.

4.2.1 Learning phase

In this phase, we distinguish *n*-gram patterns that are only expected to appear in executable contents but not in text contents, and calculate the optimal threshold for clearly distinguishing between executable and text packets. The learning phase consists of the following three steps.

1. *Identification of distinct n-gram patterns in executable and text files:*
   We identify the distinct *n*-grams of executable and text files. Table 2 shows the total number of unique *3*-grams in executable and text files (from the dataset described in the next section).
2. *Identification of* n *-gram patterns that normally appear in executable files but not in text files:*
   We deduct *n*-gram patterns of text files from those of executable files obtained in step 1. We call the set of such patterns the *Deduction model*. It is clear from the Table 2 that text contents have far fewer unique *n*-grams than executable files. Thus, there are a large number of unique *n*-gram patterns that we can normally only find in executable content.
3. *Setting a threshold:*
   We count the number of occurrences of *n*-grams in the deduction model from each executable packet, and normalize the count by dividing it by the packet size. We call this the *relative count*. We similarly compute the relative count from each text packet. We compute such relative counts from a large pool of executable and text packets.
   Suppose E is a set of relative counts from executables, and T is a set of relative counts from text. Then we find the optimal threshold value th such that th is lower than most of E and is higher than most of T (i.e., th minimizes the false negatives and positives).

4.2.2 Identification phase

Figure 6 shows the identification process for a non-multimedia packet. If the packet is identified as non-multimedia type (by the distance-based algorithm in Sect. 3), the packet payload is further processed to identify whether it is text or code-type. We compute the relative count of *n*-grams of the deduction model from the payload. If the relative count is greater than the threshold, the packet is classified as code, otherwise as text.

5 Analyzing the classification accuracy using file segments as packet payload

In this section we analyze the classification accuracy of the algorithms using file contents as packets. Specifically, we divided the sample files into 1,500-byte blocks (since the usual MTU is 1,500 bytes in Ethernet), and we assumed each block was a packet payload. We used *3*-grams in building the models, and measured the classification accuracy (the false positive and false negative rate) of the three algorithms.

5.1 Distance-based algorithm for multimedia contents

To build the executable model, we first collected 450 executable files (which are mostly from the `bin` and `system32` folders in Linux and Windows XP).

Our goal is to find the representative *3*-gram frequencies of executable files and to avoid oversampling as it may introduce noise (i.e. *n*-gram frequency that is non-representative). For this, we created 10 incremental datasets of executable files to be used in building executable models (see Table 3), and performed 10 incremental analysis as follows.

1. In the learning phase, we picked the next dataset and built the executable model.
2. In the identification phase, we compared the executable model with a set of code and multimedia packets, and computed the accuracy (we used six common multimedia-types, i.e. jpg, rm, avi, wmv, mp3 and pdf and evaluated 10,000 packets for each of the six file types and the code type).

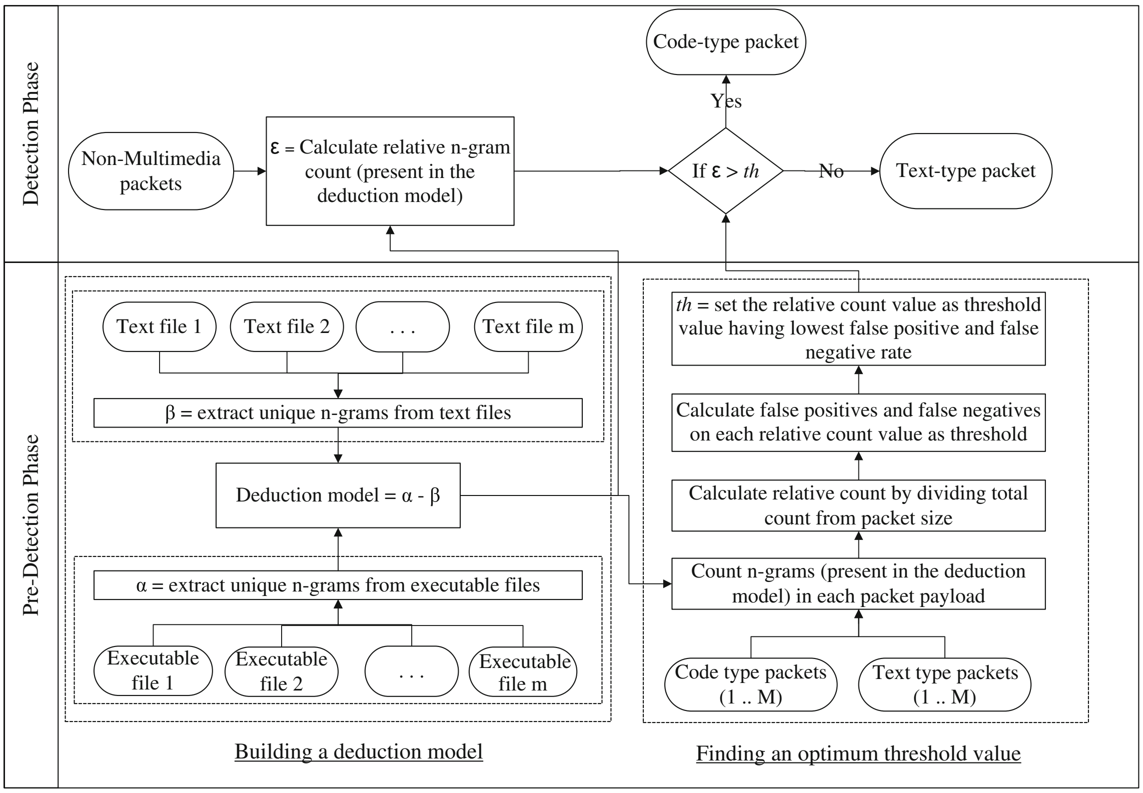

Detection Phase
Pre-Detection Phase
Code-type packet
Yes
Non-Multimedia packets
ε = Calculate relative n-gram count (present in the deduction model)
If ε > th
No
Text-type packet
Text file 1
Text file 2
. . .
Text file m
β = extract unique n-grams from text files
Deduction model = α - β
α = extract unique n-grams from executable files
Executable file 1
Executable file 2
. . .
Executable file m
th = set the relative count value as threshold value having lowest false positive and false negative rate
Calculate false positives and false negatives on each relative count value as threshold
Calculate relative count by dividing total count from packet size
Count n-grams (present in the deduction model) in each packet payload
Code type packets (1 .. M)
Text type packets (1 .. M)
Building a deduction model
Finding an optimum threshold value

Fig. 6 Pattern counting scheme

Table 3 Dataset used in building the executable models

Dataset no.	Number of program files	Total dataset size (KB)
1st	10	13,625
2nd	50	39,472
3rd	100	109,249
4th	150	167,364
5th	200	240,458
6th	250	353,573
7th	300	384,207
8th	350	391,958
9th	400	407,876
10th	450	424,427

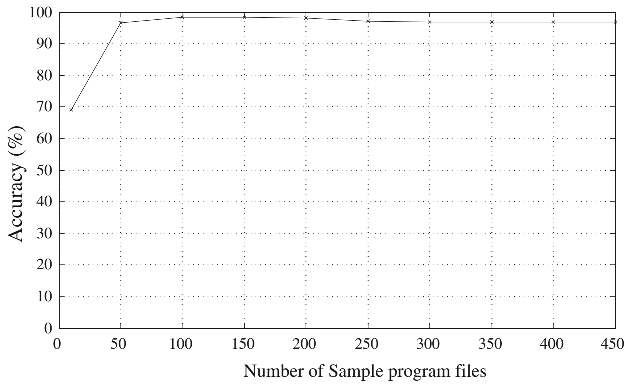


Fig. 7 Classification accuracy between code and multimedia packets, for executable models sampled by using different numbers of program files

In each analysis we calculated the accuracy as follows.

$$\text{accuracy}(\%) = \frac{(\text{true positives} + \text{truenegatives})}{\text{total number of packets}} \times 100 \tag{3}$$

Figure 7 shows the classification accuracies from the 10 analysis. We achieved the highest accuracy when the dataset containing 150 files (or about 111,000 packets) was used in building the executable model.

Table 4 shows the false negative and false positive rate of each data type, when the dataset of 150 files is used (false negatives are code packets for which the MD values are above the threshold, and false positives are multimedia packets for which the MD values are below the threshold). Most types show low false positives/negatives except for `pdf`. It is perhaps because, unlike other multimedia types, `pdf` is a compound type (that is, it contains many types of objects such as text and images).

Table 4 False positives and negatives compared with the executable model of 150 program files

File type	Rate (%)
False positives	
jpg	0.36
rm	0.54
mp3	0.37
avi	1.16
wmv	0.2
pdf	15.03
Average	2.95
False negatives	
Program files	1.44

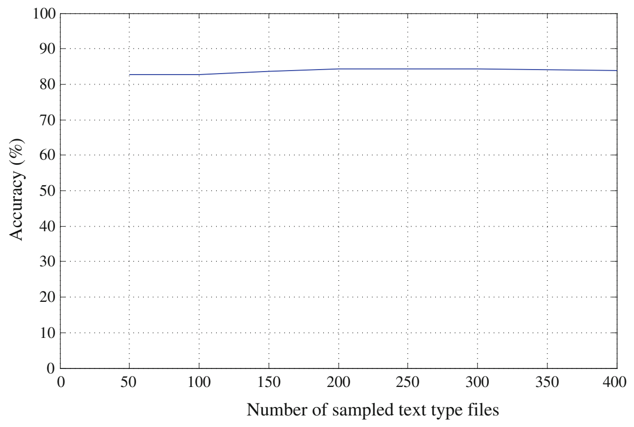


Fig. 8 Classification accuracy between code and text packets for text models sampled by using different numbers of text files

5.2 Distance-based algorithm for text contents

As we stated earlier, the distance-based algorithm for text is similar to that for multimedia contents, except that we build the model using text contents. Thus, we performed the same incremental analysis to find the optimal text model.

We randomly collected a total of 400 text files of five types (i.e. `doc`, `txt`, `xls`, `xml` and `html`) and built 10 incremental datasets of these files (see Table 5) to be used in building the text models. We also collected other types of text files (i.e. `log`, `rtf` and `asp`) which were not used in building the text model but were used in the identification phase. The purpose is to find out whether or not other text types (that are not considered when building the model) can still be (correctly) identified as text type.

In the identification phase, we compared 10,000 packets (obtained from files of the eight text types and the code type) with each of the eight models and computed their accuracies. Figure 8 shows the classification accuracies from the eight incremental analysis. We achieved the highest accuracy when the dataset containing 250 files (50 files for each type `txt`, `doc`, `xml`, `xls`, and `xml`, or about 36,000 packets total) is used in building the text model.

Table 6 False positives and negatives compared with the text model of 250 text type files

File type	Rate (%)
False positives	
asp	2.35
doc	25.67
html	0.19
log	0.78
rtf	9.72
txt	6.98
xls	20.33
xml	4.29
Average	8.79
False negatives	
Program files	22.40

Table 6 shows the details about the rate of false positives and negatives for the text model (derived from 250 sampled text files), indicating that this scheme is not accurate enough. Since different types of text files have different n-gram frequency distributions (as shown in Fig. 9),

Table 5 Dataset (total size of each file type in kilobytes, KB) used in building the text model

	Qty	TXT	DOC	XML	XLS	HTML
1st	50	328.7	4,618.6	257.9	7,670.1	116.7
2nd	100	630	6,764.8	725.8	9,375.2	309.6
3rd	150	1,540.8	11,520	1,119.6	11,516.7	341.4
4th	200	7,000.4	14,510.8	1,406.4	23,469.6	396
5th	250	7,350.2	17,972	1,728.5	26,690	430
6th	300	7,684.8	19,321.8	1,978.8	27,313.2	463.2
7th	350	13,218.8	20,351.8	2,489.9	28,703.5	546
8th	400	15,649.6	24,129.6	2,848.8	31,173.6	612

Each file type is sampled equally

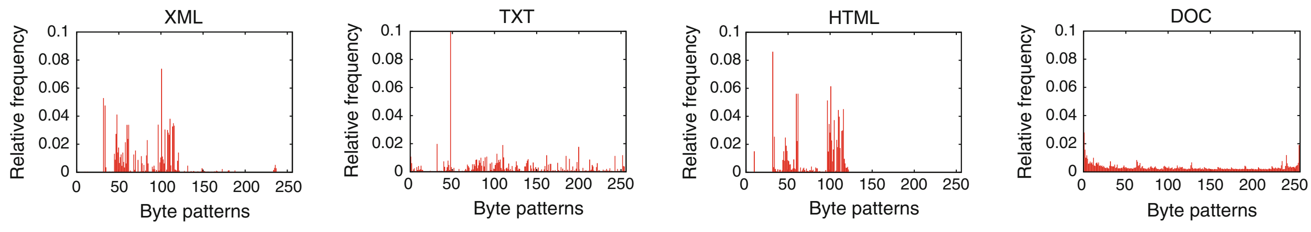


Fig. 9 *1*-gram (byte) frequency vectors of text files. They are too different to make one representative centroid model for all text file types

averaging their frequency vectors does not produce an accurate, representative text model for many types of texts; the threshold value calculated from this text model tends to be high, therefore causing a lot of false negatives (false negatives are the code packets for which the MDs are below the threshold). Moreover, the false positive rates of `doc` and `xls` are very high because, just as for `pdf`, they are of compound type (note that false positives are text packets for which the MD values are above the threshold).

5.3 Pattern counting scheme

In this experiment we used 10,000 packets from 10 text types (the same eight types as before and `jsp` and `php` in addition) and 10,000 packets from executable files. As described in Sect. 4.2, the deduction model contains distinct *n*-gram patterns that are only present in the executable files. Since executable contents have a much higher number of distinct *n*-gram patterns than text contents, code packets have a higher *relative count* than text packets. Thus, setting an optimum threshold enables us to distinguish between code and text packets.

Table 7 shows the number of false negatives and positives after calibrating the optimum threshold. Although it still shows a relatively high rate of false positives in classifying compound types such as doc and xls packets, it shows that the pattern counting scheme is far better than the distance-based algorithm in classifying text and code packets.

Table 7 False positives and negatives of pattern counting scheme

File type	Rate (%)
False positives	
asp	0.00
doc	3.10
html	0.70
jsp	0.60
log	0.00
php	0.70
rtf	0.90
txt	0.20
xls	2.40
xml	0.00
Average	0.86
False negatives	
Program files	0.38

Table 8 False positive/negative rates while protecting an FTP server

	False positives (%)	False negatives (%)
Distance-based algorithm for multimedia-type (MT)	1.65	4.00
Pattern counting scheme (PC)	0.88	0.69
Distance-based algorithm for text-type (TT)	1.99	21.89
MT + TT	3.64	26.58
MT + PC	**2.53**	**4.69**

6 Evaluating with real-world network applications and malwares

This section presents the results of experiments on real-world scenarios. Instead of using file segments as (artificial) packets and trying to classify them, we monitored real packets flowing into an ftp server and a Web browser. We also tried classifying real malwares of sizes less than 2 KB (as packets containing malwares).

6.1 Protecting an FTP server

We setup an FTP server and collected a set of text, multimedia, and code-type files to be used for this experiment. We transferred the files to the FTP server and captured the packets using Tcpdump [4]. The captured packets (70,000 in total) in the dump file were then processed using Tcptrace [5]. We used the same executable model, text model, deduction model, and threshold values we used in the previous sections.

Table 8 gives the accuracy of the distance-based algorithm for multimedia-type, the pattern counting scheme, and the distance-based algorithm for text-type. As the result shows,

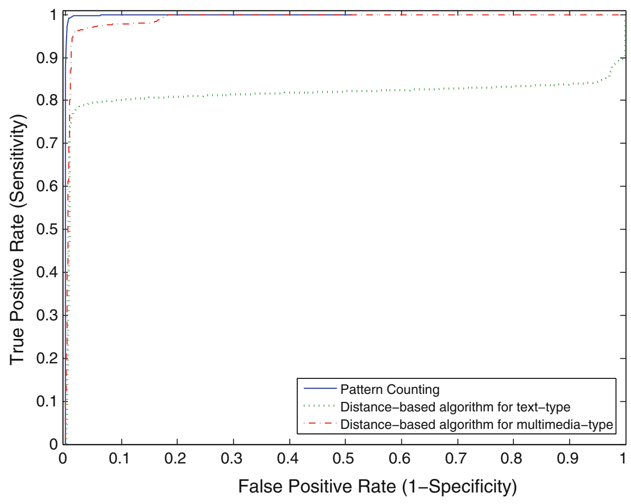


Fig. 10 ROC curve while protecting an FTP server

Table 9 False positive rate while protecting a Web client

Schemes	Total number of packets	No. of false positives	False positive rate (%)
MT + PC	35,181	1,561	4.43

the pattern counting scheme is much more accurate than the distance-based algorithm for text-type (for distinguishing text from code packets): the pattern counting scheme reduced false positives and false negatives by 56% and 97%, respectively.

Figure 10 shows the Receiver Operating Characteristic (ROC) curve for the three schemes. The result shows that the pattern counting scheme and the distance-based algorithm for multimedia-type can better classify packets.

6.2 Protecting a Web client

We captured incoming packets to a Web browser (running in Windows XP) for an hour using `tcpdump`, and processed them using `tcptrace`. The browser was used only for browsing and chatting, and we did not intentionally download activeX controls or software while capturing the packets. Thus, we were not expecting incoming executable packets.

Table 9 shows the false positive rate (there are no false negatives because there was no transmission of executable contents). The result is comparable to the previous experiment with an ftp server (2.53%). In this experiment we did not test the distance-based algorithm for text-type, which is clearly inferior to the pattern counting scheme.

6.3 Applying the content classification scheme on malwares

We tested our schemes on different types of malwares such as worms, viruses, backdoors and trojans (obtained from a publicly available Web site [2]). The sizes of malwares that we experimented on range from 64 bytes to 2Kbytes (i.e., the malwares that can more or less fit in a packet).

Table 10 False negative rate while capturing the malwares

Malware size (MS)	Total Number of malwares	No. of false negatives	False negatives rate (%)
MS $\leq$ 1 KB	25	3	12.00
1 KB $<$ MS $\leq$ 2 KB	18	3	16.66
Summary	43	6	13.95

Table 10 shows the false negative rate (using the distance-based algorithm for multimedia-type and the pattern counting scheme), which is higher (13.95%) than that of the ftp server (4.69%). However, in the next section we will discuss how to detect malwares in the face of a moderately high false negative rate.

7 Dealing with false positives and negatives

Although the proposed scheme is highly accurate, the amount of packets flowing in the network is enormous, and so is the number of false positives and negatives. In this section we discuss two ways to further enhance the proposed scheme in order to efficiently detect malwares. We present simple statistical and combinatorial analysis to identify false positives (from true positives), and to detect malwares consisting of multiple packets (in the presence of false negatives). The analysis in this section is by no means comprehensive, but we intend to argue that it is possible to devise an effective and practical detection system using our approach.

7.1 Detecting malwares in the presence of false positives

Consider a local area network based on Ethernet with a speed of 100 Mbps. The MTU size allowed by Ethernet is 1,500 bytes. Hence, the number of packets arriving per second is $(100 \times 2^{20})/(1{,}500 \times 8) = 8{,}738.13$ packets. As shown in Table 8, the overall false positive rate is 2.53 (using the distance based algorithm for multimedia-type and the pattern counting scheme). Therefore we receive approximately $8{,}738.13 \times 2.53/100 = 221.07$ false positives per second, or we can expect one false positive out of 40 packets on average. Those false positives should be further inspected to identify true positives (if any). For example, we can simply input the alarming packets to a signature detector to detect known malware.

However, we can better analyze the false positives as follows. Suppose that we have received *n* packets from a TCP session during a predefined time interval (*n* can be a multiple of 40 for convenience). Let us also assume that the occurrences of false positives are uniformly distributed over the

Table 11 Probability of identifying at least m packets of an executable file that requires n packets for its complete transfer

	$m = 1$	3	6	9	12	15	18	21
$n = 1$	0.9531000							
2	0.9978004							
3	0.9998968	0.9573961						
4	0.9999952	0.9980019						
5	0.9999998	0.9999063						
6	1	0.9999956	0.9631137					
7	1	0.9999998	0.9982700					
8	1	1	0.9999189					
9	1	1	0.9999962	0.9680640				
10	1	1	0.9999998	0.9985022				
11	1	1	1	0.9999298				
12	1	1	1	0.9999967	0.9723499			
13	1	1	1	0.9999998	0.9987032			
14	1	1	1	1	0.9999392			
15	1	1	1	1	0.9999971	0.9760607		
16	1	1	1	1	0.9999999	0.9988772		
17	1	1	1	1	1	0.9999473		
18	1	1	1	1	1	0.9999975	0.9792735	
19	1	1	1	1	1	0.9999999	0.9990279	
20	1	1	1	1	1	1	0.9999544	
21	1	1	1	1	1	1	0.9999979	0.9820551
22	1	1	1	1	1	1	0.9999999	0.9991584
23	1	1	1	1	1	1	1	0.9999605
24	1	1	1	1	1	1	1	0.9999981
25	1	1	1	1	1	1	1	0.9999999
26	1	1	1	1	1	1	1	1

interval, and the number of false positives per interval is normally distributed. If the number of alarms raised during an interval exceeds a certain threshold t, we can suspect that one or more true positives occurred in the interval, and the contents of the packets obtained during the interval are subject to further analysis that is possibly more extensive and time consuming (an example of such analysis is given in the next section). The threshold t can for example be $\mu + 2\sigma$ (where μ and σ are pre-computed mean and standard deviation), considering the empirical rule (about 95% of the values are within $\pm 2\sigma$).

This proposal enables us to systematically detect a transmission of unknown malware in the presence of false positives. The limitation of this proposal is that we do not know exactly which packets contain malware.

7.2 Detecting malwares in the presence of false negatives

This section analyzes the probability of not detecting the occurrence of a malware due to false negatives.

In Table 8, the overall false negative rate of our scheme is 4.69 (from the distance-based algorithm for multimedia-type and the pattern counting scheme). Consider a malware comprising n packets. The probability of "missing" each packet (i.e., it is not identified as a code packet) is 0.0469. Since detecting each packet is an independent event, the overall probability of missing all packets is $(0.0469)^n$. For instance, if a malware needs five packets, the probability of missing all packets is $(0.0469)^5 = 0.000022$, which is negligible.

Let $P(X = m)$ be the probability of identifying at least m packets among n packets of a malware. This can be expressed by Eq. 4.

$$P(X = m) = 1 - \sum_{i=0}^{m} (1 - 0.0469)^i (0.0469)^{n-1} \tag{4}$$

Table 11 shows the results of Eq, 4 for different values of m and n. The result shows that it is highly unlikely to miss more than two packets. Therefore, if a malware consists of

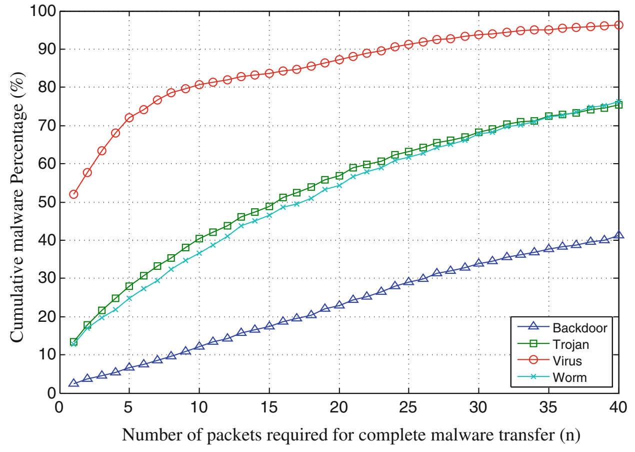


Fig. 11 Cumulative percentage of malwares that require n number of packets for their complete transfer

n packets, then at least $n - 2$ packets will be detected with more than 99.98968% probability.

Since our scheme produces about one false positive out of a stream of 40 packets (Sect. 7.1), we can detect a malware with near certainty if the malware is big enough. For example, by monitoring every stream of 40 packets of a TCP session and setting a threshold of five, we can detect malwares consisting of seven packets or more.

Since the effectiveness of our scheme depends on the size of the malware, we collected actual, publicly available malwares from VXHeaven [2] and examined their sizes. The types of malwares examined were backdoor, worm, virus and trojan. We assumed that a malware comprises $s/1500$ packets, where s is the size of the malware in bytes. Figure 11 shows the cumulative percentage of malwares. Only 8% of backdoors consist of less than seven packets. 26% and 23% of trojans and worms consist of less than seven packets, respectively. However, 70% of viruses consist of less than seven packets.

In case of a packed malware, it contains instructions that reconstruct the original code in order to execute it. When it is transmitted over network, it becomes a sequence of one or more *executable* packets followed by many *non-executable* packets. If the unpacking routine (i.e. `exe` part) is sufficiently big then our solution can directly detect it. If the unpacking routine is too small (e.g., consisting of less than 5 packets) then our solution may not detect it. However, our solution is most likely to be disturbed by the rest of the packed malware and would produce many alarms[1], which would be picked up by the statistical analysis presented in Sect. 7.1.

Further analysis of such suspected packets is out of the scope of this paper, but below we give a heuristic algorithm as an example.

[1] They are technically misclassified but in this case semantically true positives.

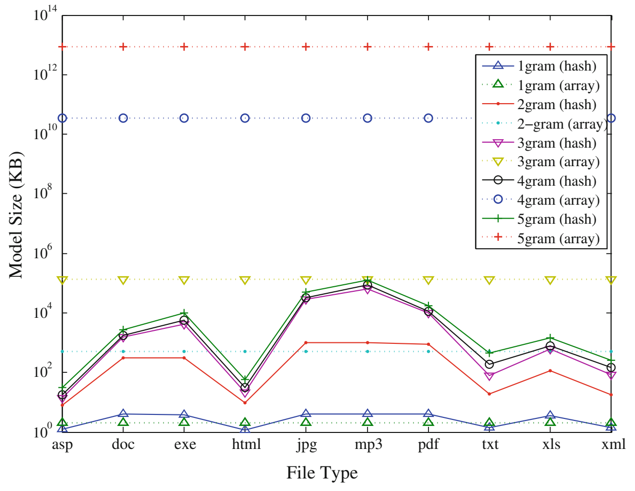


Fig. 12 Comparison between model size and file type for different orders of n-gram, using hashing and an array

- Reassemble all the packets of the interval and try matching the signatures for known archiving or packing algorithm.
- If it is found as a known archive then unzip the transmitted file and try reclassifying the unzipped files.
- If it is a known packer then we have detected a malware.
- Otherwise the alarms of the interval are indeterminable.

8 Memory requirements

Our schemes generate n-gram frequency vectors of different files and packets, which need to be stored so that frequency data can be located quickly. In this paper, we use an array and hashing and discuss their memory requirements.

In the case of an array, as the order of the n-gram increases the model size increases exponentially ($O(256^n)$). For instance, *1*-gram has 256 unique patterns so the array must have 256 entries. If eight bytes (e.g. `double` type) are required to store each frequency then our model size would be 2 KB. Although an array is easy to use, it wastes memory space when used for higher n-grams, which have many patterns that might never occur in the file or packet payload (which is the case in text types).

This problem can be overcome via hashing. In hashing, instead of allocating memory for all possible n-gram patterns at the start of the process, we allocate memory when a new n-gram pattern is found. As we discussed earlier (see Table 2), different file types have different numbers of unique n-gram patterns, hence the model size can vary with respect to the file type. Figure 12 shows the model size required by hashing and array for different orders of n-gram. The following conclusions can be drawn from the graphs.

- In an array, the model size remains the same for all file types with a given order of *n*-gram. This is because we allocated memory for all possible unique *n*-gram patterns.
- It is better to use an array for *1*-gram, because almost all possible patterns occur in all types of file and therefore nothing can be gained by using hashing. In addition, hashing needs additional data due to collision handling.
- In *2*-gram, both techniques require a similar model size for multimedia or binary file types (such as `doc`, `exe`, `jpg`, `mp3`, and `pdf`). However, hashing is more suitable for text files, because they have fewer unique *2*-gram patterns than multimedia ones.
- For *3*- and higher grams, hashing is more efficient.

9 Conclusions and future work

To detect malware, we proposed a content classification scheme that analyzes packet payload to identify executable codes. The proposed scheme consists of two steps (a distance-based classification algorithm and the pattern counting scheme), which is highly effective in detecting executable contents in packets.

We found that the Mahalanobis distance-based algorithm is highly accurate in identifying multimedia-type contents, but not text or executables. To identify text and executable contents effectively and efficiently, we proposed the pattern counting scheme.

In both the distance-based algorithm and the pattern counting scheme, we observed that the accuracy increases as the order of the *n*-gram increases. We found that using the proposed scheme with 3-gram is accurate enough to identify executable contents.

In general, anomaly detection schemes produce false negatives and false positives. Our content classification scheme is no exception, and showed 4.69% of false negatives and 2.53% of false positives, thus requires further inspection to effectively detect the occurrence of malware. However, we showed that with further statistical analysis it is possible to enhance the accuracy. An example of analyzing alarmed packets that can reduce false positives is also presented, which uses domain knowledge of known file types.

Currently, our scheme cannot achieve high accuracy in detecting compound files such as `doc, pdf` and `xls` because such a file contains multiple types of contents. As our future work, we will find a technique to effectively identify such compound files. For example, the heuristic algorithm given in Sect. 7.2 may be extended to reduce false positives of our solution such that known encoding and archiving formats such as zip, tar, or MS Office documents[2] are preprocessed to expose the contents before classification.

[2] MS Office 2007 documents are simply zipped XML files.

Acknowledgments The authors are grateful to the anonymous reviewers for their valuable feedback which has significantly improved the quality and the presentation of the paper.

References

1. Bro. http://www.bro-ids.org. Accessed 14 Nov 2010
2. Publicly available library of malwares (VX Heavens). http://vx.netlux.org/. Accessed 14 Nov 2010
3. Snort. http://www.snort.org/. Accessed 14 Nov 2010
4. Tcpdump. http://www.tcpdump.org. Accessed 14 Nov 2010
5. Tcptrace. http://www.tcptrace.org. Accessed 14 Nov 2010
6. Amirani, M.C., Toorani, M., Shirazi, A.A.B.: A new approach to content-based file type detection. In: IEEE Symposium on Computers and Communications (ISCC '08), pp. 1103–1108 (2008)
7. Bolzoni, D., Etalle, S., Hartel, P.: Poseidon: a 2-tier anomaly-based network intrusion detection system. In: Fourth IEEE International Workshop on Information Assurance (IWIA'06). London, UK (2006)
8. Calhoun, W.C., Coles, D.: Predicting the types of file fragments. Digit. Investig. **5**(1), 14–20 (2008)
9. Criscione, C., Zanero, S.: Masibty: an anomaly based intrusion prevention system for web applications. In: Black Hat Europe. Moevenpick City Center, Amsterdam, Netherlands (2009)
10. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: 16th USENIX Security Symposium, Boston, pp. 167–182 (2007)
11. Harris, R.M.: Using artificial neural networks for forensic file type identification. Technical report, Purdue University (2007)
12. Kolmogorov, A.: Three approaches to the quantitative definition of information. Problems Inf Transmission **1**(1), 1–7 (1965)
13. Kruegel, C., Toth, T., Kirda, E.: Service specific anomaly detection for network intrusion detection. In: ACM Symposium on Applied Computing, Madrid, pp. 201–208 (2010)
14. Li, W.J., Stolfo, S., Stavrou, A., Androulaki, E., Keromytis, A.D.: A study of malcode-bearing documents. In: Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Lucerne, pp. 231–250 (2007)
15. Li, W.J., Wang, K., Stolfo, S.J., Herzog, B.: Fileprints: identifying file types by *n*-gram analysis. In: Workshop on Information Assurance and Security (IAW'05), pp. 64–71. United States Military Academy, West Point, New York (2005)
16. Martin, K., Nahid, S.: File type identification of data fragments by their binary structure. In: Proceedings of the 7th Annual IEEE Information Assurance Workshop, pp. 140–147. United States Military Academy, West Point, New York (2006)
17. Martin, K., Nahid, S.: Oscar: file type identification of binary data in disk clusters and ram pages. In: Proceedings of IFIP International Information Security Conference: Security and Privacy in Dynamic Environments (SEC2006), pp. 413–424 (2006)
18. McDaniel, M., Heydari, M.H.: Content based file type detection algorithms. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, vol. 9, p. 332a (2003)
19. Shafiq, M.Z., Khayam, S.A., Farooq, M.: Embedded malware detection using markov *n*-grams. In: International Conference on Detection of Intrusions, Malware and Vulnerability Assessment (DIMVA'08), Paris, pp. 88–107 (2008)
20. Sommer, R., Paxson, V.: Enhancing byte-level network intrusion detection signatures with context. In: 10th ACM Conference on Computer and Communications Security, Washington, DC, pp. 262–271 (2003)
21. Stolfo, S.J., Wang, K., Li, W.J.: Towards stealthy malware detection. Adv. Inf. Secur. **27**, 231–249 (2007)

22. Tan, P.N., Steinbach, M., Kumar, V.: Classification: alternative techniques. In: Introduction to Data Mining. AddisonWesley, USA (2005)
23. Veenman, C.J.: Statistical disk cluster classification for file carving. In: IEEE Third International Symposium on Information Assurance and Security, pp. 393–398 (2007)
24. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: a content anomaly detector resistant to mimicry attack. In: 9th International Symposium on Recent Advances in Intrusion Detection (RAID'06), Hamburg, pp. 226–248 (2006)
25. Wang, K., Stolfo, S.J.: Anomalous payload-based network intrusion detection. In: Seventh International Symposium on Recent Advances in Intrusion Detection (RAID'04), France, pp. 203–222 (2004)
26. Wang, X., Pan, C.C., Liu, P., Zhu, S.: Sigfree: a signature-free buffer overflow attack blocker. In: 15th USENIX Security Symposium, Boston, pp. 225–240 (2006)
27. Zanero, S.: Ulisse, a network intrusion detection system. In: 4th annual workshop on cyber security and information intelligence research (CSIIRW'08). Oak Ridge, TN, USA (2008)
28. Zhang, Y., Paxson, V.: Detecting backdoors. In: 9th USENIX Security Symposium, Colorado (2000)