An Approach to Detect Executable Content for Anomaly Based Network Intrusion Detection

Like Zhang, Gregory B. White
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249

Abstract- **Since current internet threats contain not only malicious codes like Trojan or worms, but also spyware and adware which do not have explicit illegal content, it is necessary to have a mechanism to prevent hidden executable files downloading in the network traffic. In this paper, we present a new solution to identify executable content for anomaly based network intrusion detection system (NIDS) based on file byte frequency distribution. First, a brief introduction to application level anomaly detection is given, as well as some typical examples of compromising user computers by recent attacks. In addition to a review of the related research on malicious code identification and file type detection in section 2, we will also discuss the drawback when applying them for NIDS. After that, the background information of our approach is presented with examples, in which the details of how we create the profile and how to perform the detection are thoroughly discussed. The experiment results are crucial in our research because they provide the essential support for the implementing. In the final experiment simulating the situation of uploading executable files to a FTP server, our approach demonstrates great performance on the accuracy and stability.**

1. INTRODUCTION

1.1 Application Level Anomaly Detection

Internet threats can be classified into network level and application level. Network attacks, such as DOS, Arppoison, or Teardrop, usually contain no payload or meaningless payloads, and they try to exploit the packet header fields for intrusion. On the other hand, application level attacks generally do not have any malicious fields in the header portion, but in their payloads. In most cases, they aim at specific systems, such as certain version of Microsoft Windows or Linux, or specific programs, like Internet Explorer or IIS. Computer virus is a typical example of application level attacks. However, with the vast popularity of internet applications, computer virus is neither the only threat nor the most widespread one. People are facing attacks from various types including Trojan, worm, backdoor, spyware, etc., most of which are from the application level.

To defend against these threats, anti-virus software is the most used tool for general users. Anti-virus products belong to the signature-based detection approach, which identifies threats by matching known fingerprints. This method provides high accuracy for attacks already known, but not effective for zero-day attacks [1]. Zero day attacks include new type threats and variations of existing attacks which have no fingerprint at their first launches. Although patches or rescue methods will be release in short period, intense damage could have been made.

The only solution to defend against zero day attack is the anomaly based detection independent of specific signatures. The basic mechanism in the anomaly detection approach is establishing a profile to describe the “normal” situation of a network or a machine. If this profile was thorough and accurate enough, all attacks should be detected because they are “abnormal” to the profile. However, until now, there has no effective method to construct such a perfect profile. The biggest problem is the dilemma between detection rate and false positive. In the DARPA’99 experiment [2], most anomaly based NIDS products showed poor performance for U2R and R2L attacks which performed on the application level. In [3] [4] [5] [6], different recent approaches have been examined and compared. In spite that recent anomaly detection researches have adopted various

1-4244-0910-1/07/$20.00 ©2007 IEEE

techniques from data mining, neural networks, statistics and other artificial intelligence fields, no significant progress has been made for the application level threats.

Ignorance of packet payload is the major reason for the poor performance of anomaly detection on application level. In signature based methods, intense analysis was made on the packet payload to extract the unique fingerprints, so the signature-based approach could provide extremely high accuracy on existing attacks. For anomaly based methods, although many interesting algorithms have been adopted as in [2] [3] [4] [5] [6], all their efforts focus on the packet header fields only. The consequence is that these anomaly detection approaches demonstrate good accuracy for network level attacks, such as DOS, teardrop, SYN flood, etc., but not for the application level exploits, such as U2R and R2L. The reason behind this is actually fairly simple. All of these anomaly detection schemes only consider the packet header fields, such as the ip address, port number, flags, etc., so they work well when the attack involves only the related fields as most network level attacks. Once the payload is involved, these methods have no way to identify them. For example, a popular attack towards Microsoft IIS is to induce the user to download a malicious script file. Since there are no invalid packet header fields involved, those header-based approaches will not trigger any alarm. Unfortunately, malicious payload is the characteristic of application level attack. If the payload is ignored as in the traditional anomaly based intrusion detection, poor ability to identify those payload associated attacks is obvious.

There are millions of application level threats ranging from traditional computer virus to illegal requests at certain service. For regular users, the most common challenges are Internet Trojans, worms, virus, spy ware or other malicious programs. As mentioned above, current anti-virus solutions are vulnerable to zero day attacks because they are signature based, and anomaly detection lacks the reliable mechanism to construct an accurate profile to distinguish the attacks from normal events. Although it is extremely difficult to develop an effective solution for defending all unknown attacks, we found most of them have one thing in common: hidden executable content [7].

1.2 Detecting Executable Content for NIDS

In this paper, we propose an anomaly based solution to detect hidden executable content in the network traffic. This is different from detecting malicious code. Current researches on detecting malicious code focus on extracting patterns or certain models from already known attacks [8][9][10]. They can be used to prevent variants of those worms or Trojans, but cannot distinguish brand new unknown threats. Besides, sometime it is hard to tell whether a program is malicious or not. For example, *flashget* is a very popular download tool, but it also contains adware which is installed silently.

Rather than detecting the exact malicious content, we propose the method to identify any executable content as a more practical solution. Users generally do not want automatically downloading any executable file without their permission. Reputable vendors, like Microsoft or Symantec, will give prompts when they try to download patches as in their update programs, but not every software follows the rule. When a program has been installed, the system is then exposed to any future action. The installed software could silently download other files to your computer without your permission. A typical example is the 3721 Internet Assistant which is supposes to provide support for Chinese keyword search, but it became so annoying then was defined as spyware later [11]. Such programs can be anything from downloading tool to p2p software. They usually offer some extra functions like better searching, faster downloading, etc. In fact, they do provide such benefits, but secretly download other files on user machines at the same time. Regardless whether those downloaded content is hazardous or not, user will not expect such hidden events in their limited bandwidth. Especially, if the secretly downloaded content is executable, it is very possible it is a Trojan or other similar code. Based on all these situations, it is necessary to develop a solution to detect hidden executable files for NIDS as we propose in the paper.

The rest is organized as follows. Section 2 will introduce some related researches for detecting executable contents, as well as its drawbacks for NIDS. Section 3 is the introduction of the training phase of our approach. Besides the steps to construct the profile, we will focus on discussing why it is a reliable solution to use byte frequencies to identify .exe files. Section 4 presents

the details of detection phase. In section 5, experiment results are given to demonstrate the effectiveness of our mechanism.

2. RELATED RESEARCH

The purpose of our approach is to detect executable files in the network traffic by checking incoming and outgoing packet payloads. Identifying file types is nothing new. The *file* command in UNIX could check whether a file is text or executable binary file. And there are other tools on Windows platform to detect file types [12]. All these methods are following the same mechanism: unique "magic number" of a specific file type in the beginning part of the file. For instance, JPEG image files begin with 0xFFD8EF, while GIF files begin with the ASCII code "GIF89a" or "GIF87a". Compiled Java code starts with "0xCAFEBABE", while the MS-DOS format .exe file starts with 0x4D5A. Although these approaches are accurate for normal files, the file header could be obfuscated or hidden by many techniques [13]. Besides, since files are transferred by numerous packets in network traffic, these packets must be assembled first to construct the correct file header, which will delay the detection progress.

In [14], a novel approach to detect file types based on its byte distribution was proposed. The researchers introduced the idea of using byte frequency distribution to represent a specific file type. They have demonstrated that all different files share similar byte distribution if they belong to the same category (e.g. exe, pdf, ms word document, etc.). And for different file types, their byte distribution models demonstrate great variances. Thus it is possible to distinguish files of a specific type from a random set by comparing the byte distribution without using any specific signature. The experiment demonstrated an average accuracy more than 90%.

However, this approach is applied to local files only. Although the author mentioned it could also be applied to real-time network IDS, they did not perform any experiment for justification. In fact, we found that in spite the above method does have the ability to detect file types using byte distribution, it needs much more improvement for network intrusion detection in real time. The major challenge is still the fragmented file content. In [14], file byte distribution models can be constructed by the first 50 bytes, first 200 bytes, first 1000 bytes, or the whole file, and each file was compared with all models to find the closest distance. This approach does not have any problem with local file detection, but not for network traffic. First, files in the transferring phase are divided into numbers of packets and they do not always arrive in sequence. We cannot just select certain parts at our will unless the whole file has arrived. Another drawback is that we cannot compare files to all models for the most accurate result because it is too time consuming for network intrusion detection on the fly. Since our purpose is to detect the file type before it was saved to the hard drive, we need a more efficient method to identify the file category.

Our approach is proposed to solve these problems when identifying file types for NIDS, as well as some other challenges not mentioned in [14]. To provide reliable performance, it includes two phases: training phase and detection phase.

3. TRAINING PHASE

In the training phase, the correct model for executable file byte frequency distribution is established. First, we need to understand why we can use the byte frequency to identify different file types.

The whole idea of our approach is from the assumption that different file types have different byte frequency distribution models, and the distribution differences among various file types are significant enough to be used as unique identifiers. To support this assumption, we randomly collected some sets of different file types, including .exe, .doc, .pdf, .jpg, .gif (The .exe file is executable file under Microsoft Windows XP, the .doc file is the Microsoft Word document). We plot their average byte frequency distributions as in fig. 1.

It is obvious that the average byte distribution differences among all these file types are indeed significant. Thus we need to prove all individual files, if they belong to the same category, should have similar frequency distribution as well. In fig. 2, we compare the byte frequency distribution of individual files of the same category and the average (Only .exe files are listed here for demonstration. For other file types, they share the same characteristic.)

Fig. 2 showed us that the same file type does share some similar properties in their byte frequencies. However, there could

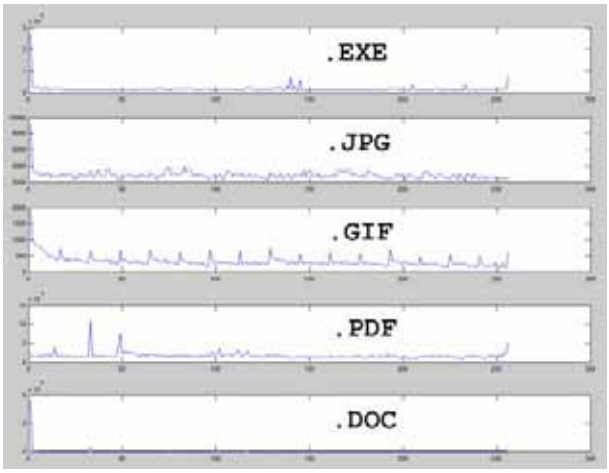


Figure 1 Average Byte Frequency Distribution of Different File Types

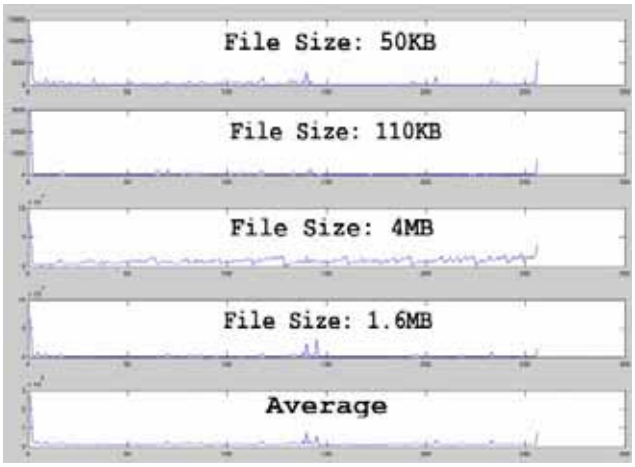


Figure 2 Individual File Byte Distribution and Average Distribution

be big variation for different file sizes. In fig. 3, we compare the byte frequencies for executable files with various file sizes.

Big file has significant difference in byte frequency distribution. The reason is that when file gets bigger, all byte frequencies increase, which change the overall result. However, from fig. 3, the frequencies increase in big files, but the frequency curve of each file is still similar (e.g. 970KB and 50KB files). We can use the following method to normalize all byte frequencies:

$$f_i = f_i - f_{\min}, i = 0,1,2,...,255 \qquad (1)$$

The f_i is the frequency for byte i, f_{min} is the minimum frequency in the distribution. Thus we could eliminate the increase amount in big files.

Another problem is that different files have different frequency ranges. In fig. 2, the maximum frequency of the .exe files varies from 10^3 to 10^6, even if the distribution curve is similar. For easier comparison in the detection phase, we have to limit the frequency to a certain range:

$$f_i = \frac{f_i}{f_{\max}}, i = 0,1,2,..., 255 \qquad (2)$$

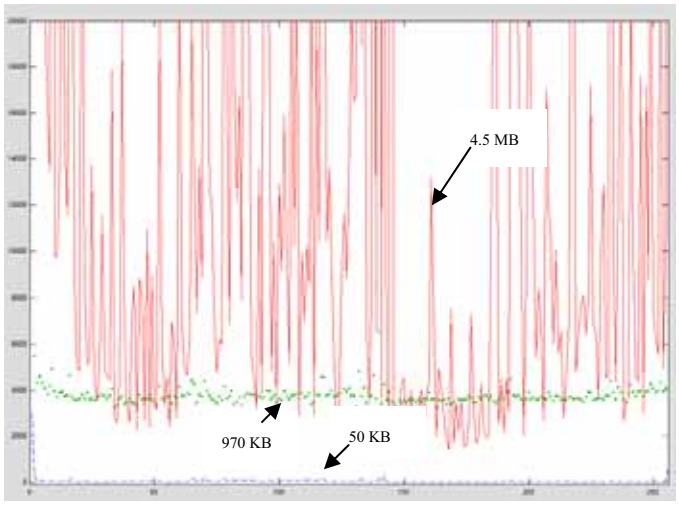


Figure 3 Compare byte frequency of big files and small files

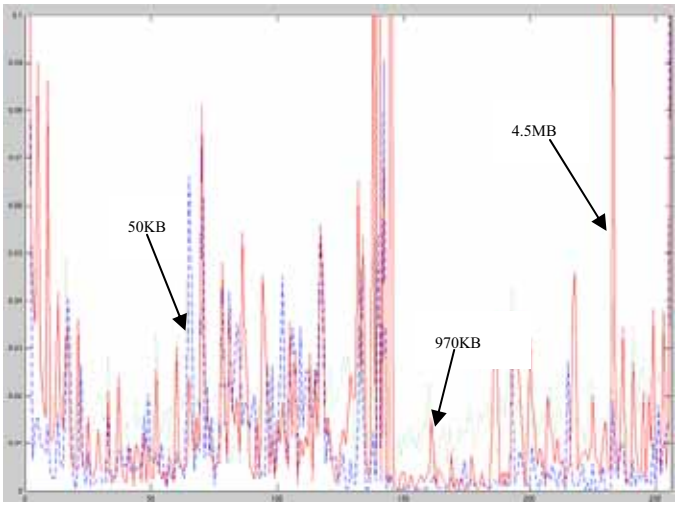


Figure 4 Normalized byte frequency distribution of different files

Thus we could map the byte frequency to the range of [0,1]. Apply the above formulas to the data in fig. 3, we have the normalized result in fig. 4.

In the training phase, we first get the total byte frequencies from all .exe files in the training set, and then apply the formula (1) (2) to normalize them. Thus we constructed a profile of the byte frequency distribution of .exe files. In the detection phase, the normalization process will be applied to the buffer which is used to store the incoming packets, then we can use Manhattan Distance to compare the buffer content with the profile, as in formula 3 ((F is the byte frequencies in the buffer, which stands for the incoming packets. M is the executable file byte frequencies profile before filter processing):

$$D = \sum_{i=0}^{i=255} | F_i - M_i | \qquad (3)$$

In our experiment, by using the normalized profile, it is already good enough to differentiate executable files from most other types including JPG, GIF, and PDF. However, when working with Microsoft Word Document (.doc files), we found the byte frequency distribution of .doc files is very close to .exe files, as in fig. 5. And this brought serious problems in the experiment.

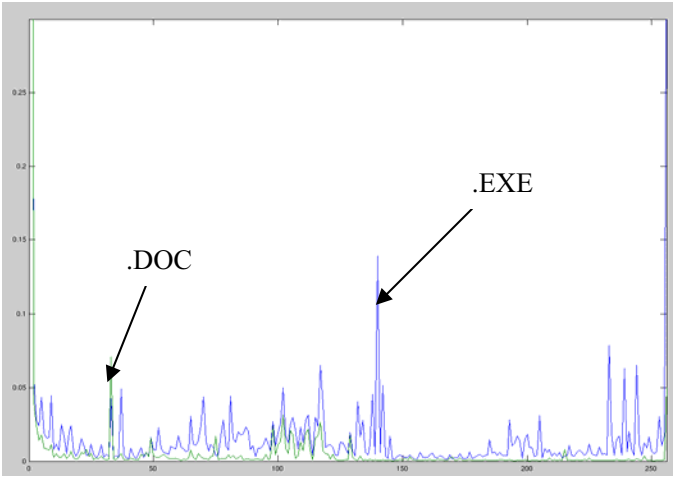


Figure 5 Average byte Frequency Distribution of .EXE files and .DOC files

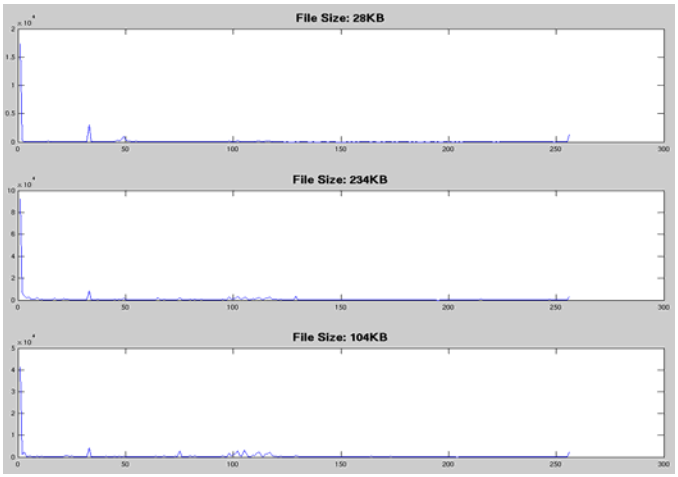


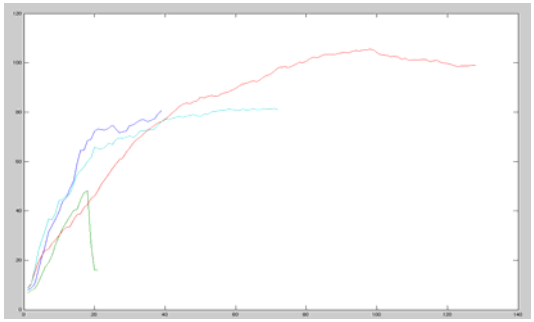
Figure 6 Byte Distribution of .DOC files

To be able to differentiate EXE files and DOC files, frequencies of individual files for both categories are examined. We found the distributions of .doc files are much stable than .exe, as in fig. 6.

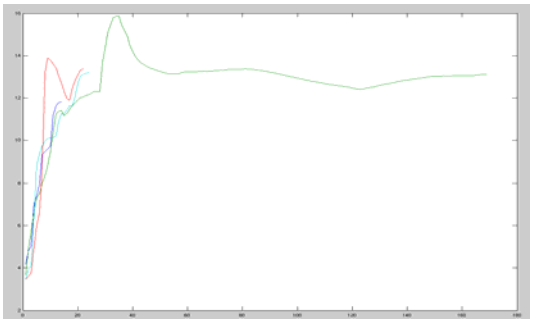
Comparing to .exe files in fig. 3, .doc files are much more stable in byte frequency distribution. Thus we could use the frequency profile of .doc files for the second round detection. Details will be described in the next section.

4. DETECTION PHASE

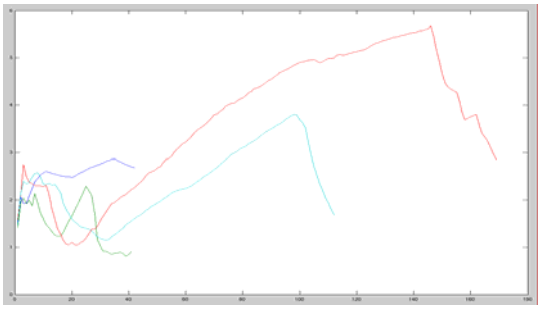
The difference between detecting different file types on local machine and in the network traffic is that we can not get the byte frequency distribution of the whole file until all packets have been received (or sent out). For individual packet of the file, no matter what kind of type it is, its byte distribution is not enough to represent the correct profile of its category. To solve this problem, we need to find out how the frequency distribution changes when continuous packets arrive. In fig. 5, different files were uploaded to a FTP server. For each upload session, a buffer is created to store the uploaded packet payload. To compare the byte frequency of accumulated packet payloads with the profile of executable files, we use the Manhattan Distance as in formula 3. The distance changes when new packets arrive have been plotted in fig. 7. For better recognition, only 4 files of each category are plotted. And we plot only 3 different categories because other types share the same characteristics.



（a）Transferring .jpg files



(b)Transferring .doc files



(c)Transferring .exe files

Figure 7　Manhattan Distance when packets arrives. The X-Axis is the packet sequence. The Y-Axis is the corresponding Manhattan Distance to the executable file profile.

From fig. 7, we found the distance of packets from .exe files always kept at a low level (<10). For packets of other types, the beginning packets might have small distance, but the distance increases so fast that will become very large within just a few more packets (usually within 5 packets). According to this, we set a buffer size to 10 packets, which means comparison of the incoming packets and the profile will be made when either the buffer is fulfilled or the file transfer is done.

As mentioned in the previous section, it is not enough to only use .exe file byte distribution profile for identification because .doc files share similar distribution with .exe files. Thus we need to perform a second detection pass using the profile of .doc file as the baseline.

The whole process of the detection phase is as fig. 8.

5. EXPERIMENT RESULTS

To test the accuracy, we created a testing set with 20 samples for each file type. There are 5 file types included: .exe, .jpg, .gif, .pdf, .doc, which are the most common file types on the internet. A FTP server is setup to simulate the situation when some users try to upload executable files to the host. Our approach works as a NIDS to detect the unauthorized executable files uploading. Besides the overall detection rate and false positive of detection, we also need to compare the results of each comparing step, as shown in table 1 and table 2. The threshold decision is based on the distance differences from the training data.

The results in table 1 and 2 demonstrated that our approach is stable and accurate for detecting executable files in network traffic. In the first round (table 1), except .doc files, all other types are eliminated since they have significant differences than our profile. The second round is to differentiate .exe files from .doc files which have similar byte frequencies. Since .doc files have a very stable frequency domain, it is much easier to compare with .doc profile rather than .exe profile only. In our experiment, when the threshold is set to 0.9, only 2 .exe files were missed, and all .doc files can be detected successfully. In order to increase the detection rate, we changed the threshold to 0.7, and all .exe files then could be identified by our algorithm.

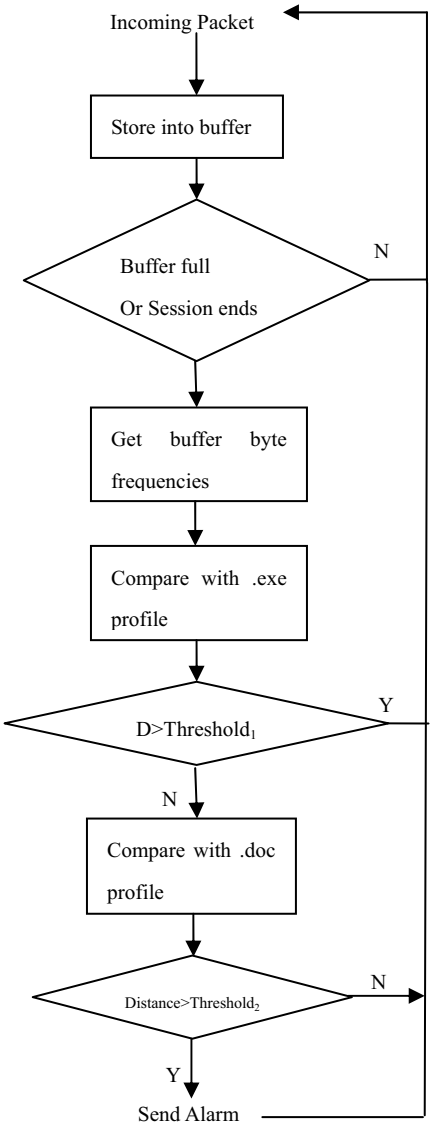


Figure 8 Detection Phase

Table 1 Result after the 1st comparison (threshold is 5)

	Detection Rate	False Positive	Min. Distance	Max. Distance	Ave. Distance
EXE	0.95	N/A	1.2351	30.5651 (3.2491)*	3.6193 (2.2011)*
JPG	N/A	0.00	8.2183	56.6764	31.7845
GIF	N/A	0.00	14.8872	43.7844	29.7862
PDF	N/A	0.00	8.5810	38.2468	17.4939
DOC	N/A	1.00	2.4514	3.3489	2.9577

* Since the only missed file is 25MB, much bigger than any file in our training set which has the maximum file size of 9MB, we consider it as not our target for detection (no Trojan or worm will be such a large file). The second data is the result excluding the big file, which is better representing the results.

Table 2 Result of after the 2nd comparison

	Detection Rate	False Positive	Min. Distance	Max. Distance	Ave. Distance
EXE	0.85 (0.90) (Threshold 0.9)	N/A	0.7011	9.3032 (3.6711)	2.2012 (1.8274)
	0.95 (1.00) (Threshold 0.7)				
DOC	1.00 (Threshold 0.9)	0.00	0.3476	0.8636	0.5676
	0.90 (Threshold 0.7)	0.10			

5. CONCLUSION

In this paper, we introduced a new approach to detect executable content in the network traffic. Although some similar research has been done for the local files for malicious code, our approach is the first towards detecting unknown executable files in the network traffic on the fly. With in depth analysis on different file categories, as well as the experiment results, we demonstrated our approach is accurate and reliable.

The future research is to introduce more data categories including video stream, audio, and compressed content to create a more realistic environment for testing the approach. It is also worthy exploring the possibility to apply similar techniques on encrypted network traffic which could be beneficial for IPv6 network.

6. REFERENCES

[1] Levy, E., "Approaching Zero", *IEEE Security & Privacy Magazine, vol. 2, issue 4, pp. 65-66*, 2004

[2] R. Lippmann, et al., *"The 1999 DARPA Off-Line Intrusion Detection Evaluation", Computer Networks, 34(4), pp. 579-595*, 2000

[3] Lazarevic, A., Ertoz, L., Ozgur, A, Srivastava, J., Kumar, V., "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection", *Proceedings of the 3rd SIAM Conference on Data Mining*, San Francisco, May, 2003

[4] Wenke Lee, Sal Stolfo, and Kui Mok., "A Data Mining Framework for Building Intrusion Detection Models", *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1999

[5] Wenjie Hu, Yihua Liao, V. Rao Vemuri, "Robust Support Vector Machines for Anomaly Detection in Computer Security", *International Conference on Machine Learning*, Los Angeles, CA, July, 2003

[6] Khaled Labib and V. Rao Vemuri, "An Application of Principal Component Analysis to the Detection and Visualization of Computer Network Attacks", *Annals of Telecommunications, France*. Nov/Dec 2005 Issue

[7] "What you should know about Download.ject", http://www.microsoft.com/security/incident/download_ject.mspx,

[8] Mihai Christodorescu, Somesh Jha, "Static Analysis of Executables to Detect Malicious Patterns", the Proceedings of the 12th USENIX Security Symposium (Security'03), pp. 169-186, August, 2003

[9] J. Kinder, S. Katzenbeisser, C. Schallhart,
H. Veith, "Detecting Malicious Code by Model Checking", Conference on Detection of Intrusions and Malware & Vulnerability Assessment, DIMVA 2005

[10] Tony A. Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "Detection of New Malicious Code Using N-grams Signatures", *the 2nd Annual Conference on Privacy, Security and Trust*, October, 2004

[11] 3721 Internet Assistant, http://en.wikipedia.org/wiki/3721

[12] FileAlyzer, http://www.safer-networking.org/en/ filealyzer

[13] P. Szor and P. Ferrie, "Hunting for Metamorphic", In *Proceedings of Virus Bulletin Conference*, pp. 123-144, 2001

[14] W.J.Li, K.Wang, S.J.Stolfo, B.Herzog, "Fileprints: Identifying File Types by N-gram Analysis", Proceedings of the 2005 IEEE Workshop on Information Assurance, 2005