情報保護學會論文誌
第 21 卷 第 3 號, 2011. 6

SVM을 이용한 HTTP 터널링 검출*

하 등 과[1†], 양 대 헌[1], 이 경 희[2‡]
[1]인하대학교, [2]수원대학교

Detect HTTP Tunnels Using Support Vector Machines*

Dengke He,[1†] DaeHun Nyang[1], KyungHee Lee[2‡]
[1]INHA University, [2]University of Suwon

요 약

최근 모든 네트워크에서 사용자가 웹 페이지에 접근할 때 HTTP가 폭넓게 사용되기 때문에 HTTP 트래픽은 방화벽과 다른 게이트웨이 보안 장치를 통과할 때 보통 별도의 검사 절차 없이 로컬 보안 정책에 의해서 통과된다. 하지만 이러한 특성은 악의적인 사람에 의해 사용될 수 있다. HTTP 터널 응용 프로그램의 도움으로 악의적인 사람은 로컬 보안 정책을 우회하기 위해 HTTP로 데이터를 전송할 수 있다. 따라서 보통의 HTTP 트래픽과 터널링된 HTTP 트래픽을 구별하는 것은 아주 중요하다. 우리는 터널링된 HTTP 트래픽을 검출하는 통계적인 방법을 제안한다. 우리가 제안하는 방법은 사이트 독립적이기 때문에 지역적 제약을 갖지 않는다. 우리가 제안한 방법은 한 번의 학습만으로도 다른 사이트에 적용될 수 있다. 게다가 우리의 방법은 높은 정확도로 터널링된 HTTP 트래픽을 검출할 수 있다.

ABSTRACT

Hyper Text Transfer Protocol(HTTP) is widely used in nearly every network when people access web pages, therefore HTTP traffic is usually allowed by local security policies to pass though firewalls and other gateway security devices without examination. However this characteristic can be used by malicious people. With the help of HTTP tunnel applications, malicious people can transmit data within HTTP in order to circumvent local security policies. Thus it is quite important to distinguish between regular HTTP traffic and tunneled HTTP traffic. Our work of HTTP tunnel detection is based on Support Vector Machines. The experimental results show the high accuracy of HTTP tunnel detection. Moreover, being trained once, our work of HTTP tunnel detection can be applied to other places without training any more.

Keywords: HTTP tunnels, site independent, traffic classification, SVM

I. Introduction

Modern network security environment is like this: a firewall is installed in the network boundary checking network connections. If the port of the connection is not allowed by network administrator, then the firewall will block the network connection targeting on this port. In a network, an Intrusion Detection System (IDS) is probably used to detect possible attacks by checking the payloads of network packets. If one packet payload contains the signature matching IDS rules, then alert or action might be taken.

Normally web browsing is a common be-

접수일(2010년 8월 19일), 수정일(2011년 3월 9일), 게재확정일(2011년 4월 27일)
이 논문은 인하대학교의 지원에 의하여 연구되었음.
† 주저자, dengke.he@inha.edu
‡ 교신저자, khlee@suwon.ac.kr

havior within a local network, which is allowed by network administrators. HTTP is the protocol widely used for web browsing. Hypertext Transfer Protocol Secure (HTTPS) is also used for browsing web pages, but so few web pages need HTTPS in comparison with HTTP. Therefore, HTTP is normally allowed by firewall. IDS may also checks the payload of HTTP traffic to see if there is something abnormal.

This security environment however, can be easily broken by several techniques. For example, file sharing applications (eMule/eDonkey etc.) disguise their traffic as HTTP traffic in order to pass the examination of firewall and IDS. Tunnel technique is another big thread to this security environment. DNS tunnel[1] and HTTP tunnel[2][3] are two typical tunnel techniques making use of the character of this security environment. By encapsulating application data into protocols allowed by security policies, any applications using tunnel techniques could work properly even if they are actually forbidden by the security policies.

However, HTTP tunnels could be recognized using statistical methods because regular HTTP traffic have different statistical characteristics with tunneled traffic. Suppose a user is chatting using tunneled instant messaging. The chatting behavior, which includes sending short messages and receiving short messages is quite different from web browsing, which includes sending short messages and receiving long messages.

Since applications have different behaviors, we proposed a HTTP tunnel detection method based on statistical mechanism. The proposed method is site independent, which means there is only one training time. Once being trained, this method can be applied to any other sites without further training. Besides, the proposed method achieves a high accuracy, mostly 99%.

The rest of this paper is organized as follows. Related work are discussed in Section 2, then in Section 3 we describe the datasets we used. In Section 4 we describe how SVM works and how we set our classification parameters. Detailed classification process are presented in Section 5 and classification results are discussed in Section 6. Finally we conclude in Section 7.

II. Related Work

Serving a very long time historically, port based IP traffic classification is based on registered port numbers in Internet Assigned Numbers Authority (IANA)[4]. IP traffic are classified by their port numbers into different applications, e.g. port 80 for HTTP. This classification method is simple but quite unreliable. The reasons are that not every application has its registered port, and not every application follows its registered port. For example peer to peer applications may use port 80 to transmit data. Madhukar and Williamson[5] observed that port based classification is not able to identify 30-70% of Internet traffic they investigated. This method is unable to detect HTTP tunnels since HTTP tunnels are using HTTP port as the transmit port.

To overcome the flaw of port based traffic classification, payload based IP traffic classification was proposed. Payload based IP traffic classification inspects payload of traffic to see if there is a match between examined payload and its own signature database. Many IDS such as Bro[6][7] and Snort[8] are using this approach. Several research works also used this approach[9][10][11][12]. Since payload based IP traffic classification depends on its signature database, the biggest disadvantage is that it should maintain an up-to-date

signature database, otherwise its classification ability will decrease. Another problem payload based IP traffic classification meets is that it must inspect the traffic payload which is against users privacy. In some areas, inspecting payload is against the law. The third problem of payload based IP traffic classification is that it needs high computational power since it performs deep inspection of large number of network traffic. This classification method could detect some HTTP tunnels using plain text, such as Telnet, but its detection on encrypted HTTP tunnels, such as instant messaging is doubtable.

Statistical based traffic classification is an alternative to payload based traffic classification. This approach relies on statistical properties of network traffic. It is believed that applications have different characteristics in network traffic features, such as flow duration, packet size, inter-arrival time, etc. By analysis these features statistically, different applications could be classified. This approach needs training phase and classification phase. In training phase, training samples are used to feed the classification algorithm in order to establish the classification model. As soon as the classification model is established, the classification can be performed. Paxson[13] noticed the relationship between the class of traffic and its observed statistical properties. Roughan et al.[14] proposed to map different network applications to predetermined QoS traffic classes using the Nearest Neighbors (NN), Linear Discriminate Analysis (LDA) and Quadratic Discriminant Analysis (QDA) algorithms. Erman et al.[15] presented a semisupervised traffic classification method which combines supervised and unsupervised approaches. In 2007 Erman et al.[16] proposed an approach to identify Web and peer-to-peer traffic of network core using K-means. Their research is mainly about how to distinguish HTTP traffic and peer-to-peer traffic, not targeting on HTTP tunnels. S. Kaoprakhon and V. Visoottiviseth[17] proposed a combination of signature based and behavior based approach to distinguish between normal HTTP traffic and audio/video traffic. They got a good experimental results, but still not aiming at HTTP tunnels.

M. Crotti et al.[18] presented a statistical fingerprints approach to distinguish between HTTP and HTTP tunnels. But their approach faces the problem of site dependence, which means if their method moves to another place, it should be trained again in order to adapt to local environment. Their fingerprints method works in this way:

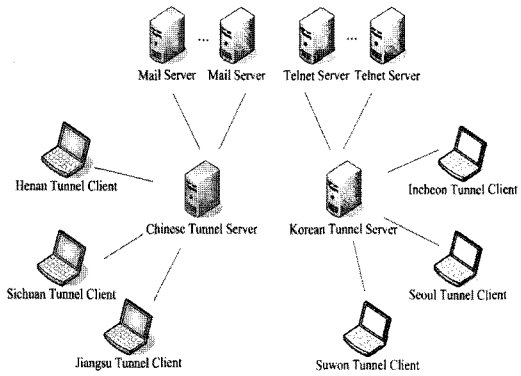
They extracted packet size, packet order and inter-arrival time as classification features to build a vector of Probability Density Functions (PDF). These PDFs are called Protocol Fingerprints. Elements in PDF are $\{s_i, \Delta t_i\}$ pairs, where s_i is the size of the *i-th* packet and Δt_i represents the inter-arrival time between the *i-th* packet and the preceding one.

During the training phase, such PDFs are established. While in the classification phase, they computed the probability of each packet in one session falling on the popular area of the fingerprints. If every packet falls on the popular regions of each PDFs, then they consider this session is normal. Otherwise it is HTTP tunnel session.

But this method has location restriction. Their other works on tunnel detection[19][20] face the same problem.

III. Date Collection

In order to prove our method is site in-



(Figure 1) Data Collection Environment

dependent on HTTP tunnel detection, we collected data from six different places. The collection environment is like (Figure 1).

There are two tunnel servers in our experiments, one in each country, China and Korea. We used computers with public IP addresses as tunnel servers. We tried to simulate the real tunnel environment. Suppose there is a user who wants to tunnel his mail data in HTTP from his company, he would like to set his home computer as the tunnel server which is normally not far from his company. In other words, his tunnel server is in his own country. Thus, we set two tunnel servers and let each country's tunnel clients connect to local tunnel servers.

There are six tunnel clients, three in China, Henan Province (CHHEN), Sichuan Province (CHSIC) and Jiangsu Province (CHJIA). The other three tunnel clients are in Korea, Incheon City (KOINC), Seoul City (KOSEO) and Suwon City (KOSUW). We let friends in these six locations run HTTP tunnel client program on their computers. Tunnel clients connect with tunnel servers

(Table 1) Training Datasets

Location	Item	Size (MB)	Packets	Sessions
KOINC	HTTP	292	439,022	15,818
	Tunneled SMTP	168	479,808	1,831
	Tunneled POP	52	153,049	871
	Tunneled Telnet	12	91,211	93

(Table 2) Test Datasets

Location	Item	Size (MB)	Packets	Sessions
KOINC	HTTP	306	435,772	15,781
	Tunneled SMTP	30	107,966	478
	Tunneled POP	34	146,574	1,078
	Tunneled Telnet	23	172,935	245
KOSEO	HTTP	311	429,184	8,925
	Tunneled SMTP	68	214,753	1,831
	Tunneled POP	66	186,979	1,441
	Tunneled Telnet	3	29,971	93
KOSUW	HTTP	114	179,781	4,929
	Tunneled SMTP	56	240,610	1,142
	Tunneled POP	59	238,533	1,537
	Tunneled Telnet	16	98,063	200
CHHEN	HTTP	446	117,659	3,883
	Tunneled SMTP	44	194,164	824
	Tunneled POP	66	42,682	1,281
	Tunneled Telnet	21	138,351	192
CHSIC	HTTP	291	435,857	15,793
	Tunneled SMTP	30	119,802	642
	Tunneled POP	28	97,363	590
	Tunneled Telnet	13	87,388	118
CHJIA	HTTP	225	366,660	2,975
	Tunneled SMTP	84	370,282	1,293
	Tunneled POP	80	273,699	1,587
	Tunneled Telnet	13	86,025	155

that will connect with the real mail servers and telnet servers when receiving data request from tunnel clients.

During one week, our friends in six locations accessed HTTP web pages, sent and received mails using different mail accounts and obtained telnet service from different telnet servers through tunneled HTTP. At the same time, an open network flow recording tool Wireshark[21] was running on their computers to capture HTTP data as well as tunneled HTTP data.

In order to guarantee that we collected the real HTTP data and the real tunneled HTTP data, we let our friends do the experiments manually. For example, to collect HTTP data, they first started Wireshark, set capture filter to 'port 80', then started web browser to access web pages. Moreover, we used different ports for different tunneled service, e.g. port 2500 for SMTP tunnel, port 11000 for POP tunnel, so we know if we capture traffic in these ports, we will get pure tunneled HTTP data.

These data provide our training datasets and test datasets, as shows in (Table 1) and (Table 2). Our training datasets are

from Incheon, Korea (KOINC), having 292 MB HTTP data, 439,022 packets and 15,818 sessions. Tunneled SMTP data are 168 MB, 479,808 packets and 1,831 sessions. There are 52 MB, 153,049 packets, 871 sessions, and 12 MB, 91,211 packets, 93 sessions for tunneled POP and tunneled Telnet respectively.

There are two terms should be explained here: packet and session. When we say packet, it is a TCP packet with Ethernet part, IP part, TCP part and maybe payload. A session is a transmission unit starting by the TCP three-way handshake ending by the FIN or RST packets. In our experiment, we tested how many sessions were correctly recognized.

IV. Support Vector Machines

Support Vector Machines (SVM) are a set of related supervised learning methods which analyze data and recognize patterns, used for statistical classification and regression analysis. Since an SVM is a classifier, then given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. Intuitively, an SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on[22].

Given a training set of instance-label pairs $(x_i, y_i), i=1,...,l$ where $x_i \in R^n$ and $y \in 1,-1^l$, SVMs require the solution of the following optimization problem:

$$\min_{(w,b,\xi)} \frac{1}{2} w^T w + C\sum_{i=1}^{l} \xi_i$$

subject to

$$y_i(w^T\Phi(x_i)+b) \geq 1-\xi_i, \xi_i \geq 0$$

Here training vectors x_i are mapped into a higher (maybe infinite) dimensional space by the function Φ. SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. $C>0$ is the penalty parameter of the error term. Furthermore, $K(x_i, x_j) \equiv \Phi(x_i)^T\Phi(x_i)$ is called the kernel function. There are four basic kernels as below:

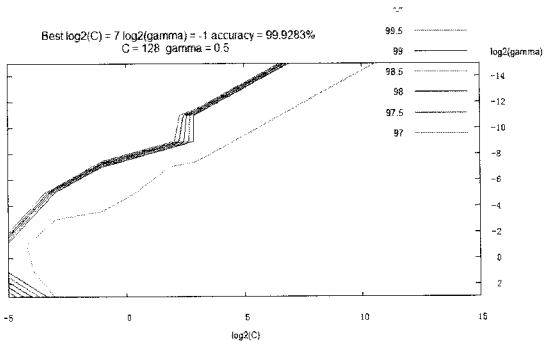
- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$
- Radial Basis Function (RBF): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0$
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

Here, γ, r, and d are kernel parameters.

In our experiment, we used Radial Basis Function (RBF) as kernel because of its good performance shown in different applications. When using RBF kernel, two parameters C and γ must be carefully chosen, as SVM classification accuracy depends on these two parameters.

We used libsvm[23] to get the best (C, γ) values as (128, 0.5) with prediction accuracy of 99.9283% on training dataset KOINC as shows in [Figure 2].

Therefore in the following experiments, we also used parameters $C=128$ and $\gamma=0.5$.



[Figure 2] Kernel Function Parameters Selection

V. Classification of HTTP Tunnels

5.1 HTTP Tunnel Mechanism

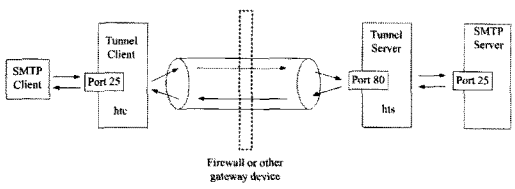
HTTP tunnel is a technique that other protocols are wrapped up by HTTP in order to circumvent local security policies.

Some networks define strict access policies to enhance local network security, such as block port 1863 to limit instant chat, block port 25 to limit mail sending, etc. While in most networks web browsing is allowed, which means port 80 is not blocked by security policy. In order to use other applications that are blocked by local security policies, HTTP tunnel was invented.

Data of other applications could be wrapped up by HTTP to disguise as normal HTTP data. By doing this, these data could pass examination of security devices usually locating in the boundary of the network.

HTTP tunnel technique normally contains two parts, tunnel client and tunnel server. The tunnel client wraps application data into HTTP data, and then sends data to the tunnel server. The tunnel server unwraps the received HTTP data into normal application data and then sends the data to the real destination. The tunnel server acts as a proxy server between the application and the real destination. When the tunnel server receives any data from the real destination, it does the similar thing as the tunnel client.

[Figure 3] shows an example of GNU HTTP tunnel[2] for SMTP. The following instructions might establish such tunnel:



[Figure 3] GNU HTTP Tunnel for SMTP

htc.exe -F 25 tunnel_server:80
hts.exe -F smtp_server:25 80

The first instruction sets the listening port (port 25) of the tunnel client and its forwarding address (tunnel_server:80). Any data received from port 25 will be forwarded to tunnel_server:80. The second instruction tells the tunnel server to listen on port 80, and forward data to smtp_server:25.

The tunnel client *htc* listens on TCP port 25 which is used to communicate with SMTP client. Any data from SMTP client will be encapsulated into HTTP data by tunnel client, and then send to tunnel server, i.e. tunnel_server:80.

Since the encapsulated data are much like regular HTTP data at first glance: with HTTP 'GET'/'POST' request, with HTTP response, and using port 80, firewall or other gateway security devices would let them pass according to the policies.

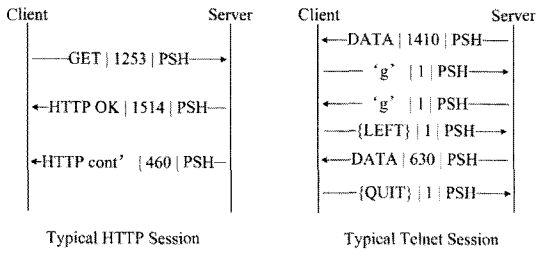
When the tunnel server *hts* receives the encapsulated data, it performs decapsulation to get the original SMTP data. Then it communicates with the real SMTP server. Any data from real SMTP server would be wrapped up into HTTP data and then sent back through the tunnel between the tunnel client and the tunnel server.

After the tunnel client gets data from the tunnel, it decapsulates the data and sends to SMTP client.

Usually the tunnel client *htc* runs on local user's machine, the tunnel server *hts* runs on a machine outside the local network, a computer with a public IP address at home, for example.

5.2 Classification Features Selection

Our classification features are chosen based on the observation of [Figure 4], the difference between a typical HTTP session



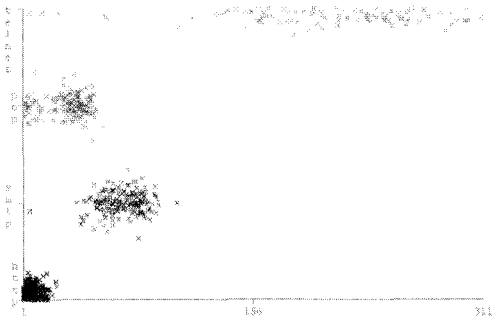
[Figure 4] Typical Sessions of HTTP and Telnet

and a Telnet session. In [Figure 4] we omit the control packets, such as SYN, ACK, FIN, RST packets etc., because they are related to TCP transmission instead of application. In our method, we only consider packets with payload.

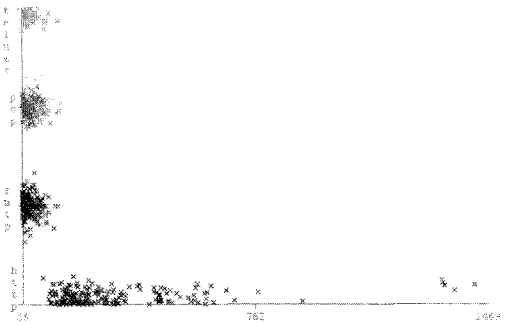
In a typical HTTP session, the client first sends a HTTP 'GET' or 'POST' request packet, usually with a large payload, more than 100 bytes and with a PUSH flag. According to TCP RFC[24], the PUSH flag of a packet indicates that the receiver must not wait for more data from the sender and process the buffered data immediately. After receiving the client's request, the server will respond with requested HTTP data. Normally several large packets with PUSH flags will be sent back to the client.

In a typical Telnet session, the server first sends welcome information to the client, then the client authenticates himself with user name and password. Every character the client types will send back to the client in order to display on the client's screen. After authenticate successfully the client sends commands to the server, and gets responses. Normally the response packets from the server are larger than the command packets from the client. Most of the packets in a Telnet session have PUSH flags.

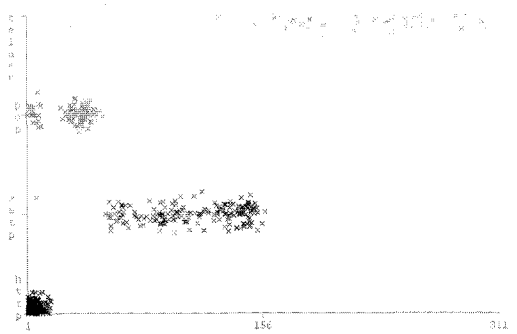
Through above analysis we can find that a HTTP session is quite different from a Telnet session in the numbers of PUSH packets, packet size and packet numbers.



[Figure 5] Number of PUSH Packets from Client to Server



[Figure 6(a)] Minimal Packet Length from Client to Server



[Figure 6(b)] Packet Numbers from Client to Server

The similar situation happens to SMTP and POP sessions. We could also demonstrate this point through 2D scatter plot graphs ([Figure 5] and [Figure 6]). These graphs are from KOINC training data.

From [Figure 5] we could see that PUSH numbers of normal HTTP session are scattered in the area of (1, 30). For instance,

one HTTP session has 20 PUSH packets, while another HTTP session may has just 2 PUSH packets. In overall, these PUSH numbers are probably confined in the area of 1 to 30.

POP data are mainly scattered in (30, 60), and SMTP in (60, 80). While Telnet are spread between 140 and 311.

[Figure 6(a)] and [Figure 6(b)] show the scattered plots of the minimal packets length and packet numbers respectively. From these figures we could also find that HTTP has differences in packet length and numbers with HTTP tunnels.

Therefore, the number of PUSH packets, packet length and packet numbers of both client/server and server/client directions are selected as our classification features. [Table 3] shows the total 16 features we used in our method.

5.3 Classifier Accuracy Comparison

Except SVM, we also tried three other classifiers: ZeroR, Naive Bayes and AdaBoost on test datasets of Incheon City, Korea (KOINC).

ZeroR or 0-R classifier belongs to rule classifier. In rule classifier, different rules are applied to different attributes, and based on these rules an output is chosen. The ZeroR classifier takes a look at the target attribute and will always output the value that is commonly found in that column[25]. It predicts the test data's majority class (if nominal) or average value (if numeric)[26].

[Table 3] Classification Features

ID	Feature	Meaning
1	c.s_push_numbers	Number of push packets from client to server
2	s.c_push_numbers	Number of push packets from server to client
3	c.s_push_ratio	Ratio of c.s_push_numbers and s.c_push_numbers
4	c.s_packet_numbers	Number of packets from client to server
5	s.c_packet_numbers	Number of packets from server to client
6	c.s_packets_ratio	Ratio of c.s_packet_numbers and s.c_packet_numbers
7	c.s_min_packet_length	The minimal packet length from client to server
8	c.s_max_packet_length	The maximal packet length from client to server
9	c.s_mean_packet_length	The mean packet length from client to server
10	c.s_packet_length_stddev	The standard deviation of packet length from client to server
11	c.s_total_length	The total packet length from client to server
12	s.c_min_packet_length	The minimal packet length from server to client
13	s.c_max_packet_length	The maximal packet length from server to client
14	s.c_mean_packet_length	The mean packet length from server to client
15	s.c_packet_length_stddev	The standard deviation of packet length from server to client
16	s.c_total_length	The total packet length from server to client

[Table 4] Classifier Accuracy and Training Time

Classifier	ZeroR	Naive Bayes	SVM	AdaBoost
Accuracy	28.99%	95.36%	97.97%	57.97%
Training Time	0.23s	0.12s	0.15s	0.31s

Naive Bayes is based on applying Bayes' theorem with strong independence assumptions. It assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. This classification method analyzes the relationship between instance of each class and each attributes to get a conditional probability for the relationship between the attributes and the class. An advantage of Naive Bayes classifier is that it requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. More information can be found in [27].

AdaBoost or Adaptive Boosting is a meta-algorithm which be used in conjunction with many other learning algorithms to improve their performance. It incrementally constructs a complex classifier by overlapping the performance of possibly hundreds of simple classifiers using a voting scheme. AdaBoost is simple to implement and known to work well on very large sets of features by selecting the features required for good classification[28]. Detailed information could be found in[29].

The experiment result shows as [Table 4]. We found that ZeroR has the poorest performance, only has the accuracy of 28.99%. While SVM achieves the accuracy of 97.97%, the best performance among all of them. Besides, SVM and Naive Bayes have the shortest training time periods.

However in comparison with the accuracy performance, the 0.03 seconds' difference between SVM and Naive Bayes can be ignored. Therefore, SVM was chosen as the classifier of our method.

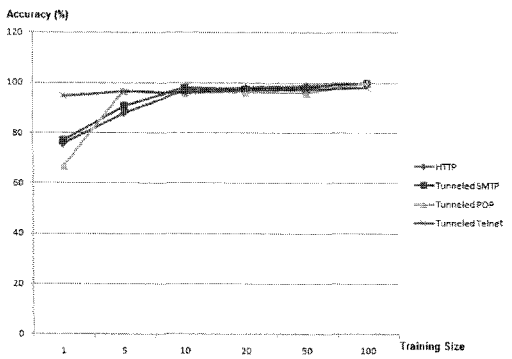
5.4 Training Size

We also conducted the experiment on how many samples needed for SVM to get a high accuracy. [Figure 7] shows the results.

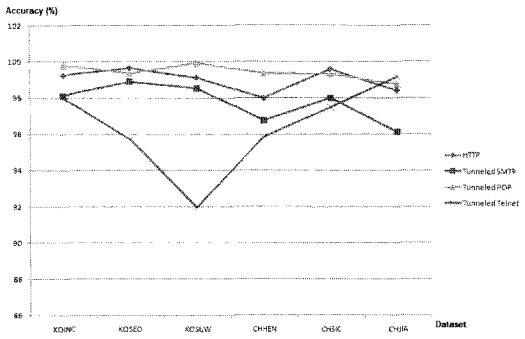
From [Figure 7] we can see that using SVM algorithm can achieve an acceptable results (about 96%) with 10 training samples, i.e. 10 sessions from training datasets and a very promising results (almost 100%) with 100 training samples. Thus, we chose 100 training samples in the following experiments.

VI. Experimental Results

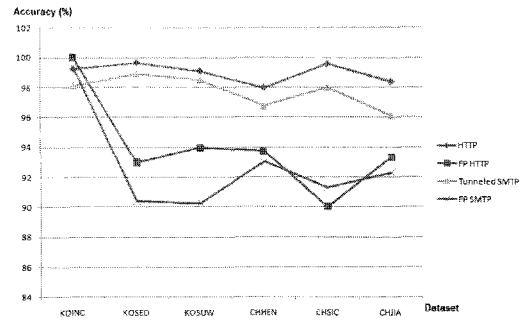
We tested six datasets from China and Korea and got the results as showed in [Figure 8]. Most of them are quite accurate, 99%, only one is below 95%. The training dataset are from Incheon, Korea, but the test datasets are from different locations of Korea and China. We even tried to use KOSEO and CHSIC as our training dataset and tested five other datasets, similar results have been gotten.



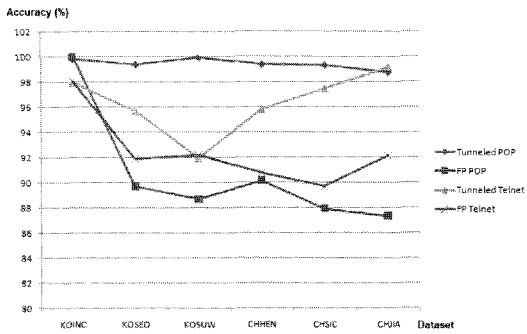
[Figure 7] Accuracy of Different Training Size



[Figure 8] Experimental Results



[Figure 9] Comparison Results: HTTP and SMTP



[Figure 10] Comparision Results: POP and Telnet

Therefore we conclude that our proposed method can detect HTTP tunnels without location restrictions, that means it is site independent. Being trained once, the proposed method could work on other sites with high accuracy.

We also compared our experimental results with M. Crotti et al.'s Protocol Fingerprints method (FP)[18] in [Figure 9] and [Figure 10].

[Figure 9] shows the comparison results between our HTTP accuracy and FP accuracy, as well as SMTP comparison. From this figure we can see that when dealing with dataset KOINC, both methods perform quite well, achieving the accuracy about 100%. This is because the training dataset and classification dataset are from the same place, Incheon, Korea. It is coincident with the experimental results of FP method[18].

When we tested both methods with datasets from different places, differences illustrated in [Figure 9]. Both HTTP and SMTP using FP method achieve the lower accuracy in comparison with our method with datasets from five other places. FP accuracy using the five datasets besides KOINC are around 92%, while our accuracy are around 98%. This confirms that FP method is location restricted and our method is not restricted to single location.

The same situation happened when we tested POP and Telnet using both methods. The comparison results show in [Figure 10]. Being trained and tested with dataset from same place, both methods get good results. Being tested with different datasets, our method is superior to FP method, except one point for Telnet in KOSUW.

VII. Conclusion

We proposed a HTTP tunnel detection method based on statistical mechanism. We did experiments to train our method with datasets from one location and test our method with six different locations from two nations. In comparison with the existing methods, the experimental results showed that our proposed method is site independent. It only needs one training time and could be applied to other networks without training any more. Besides, the experimental results showed that our proposed method achieves high accuracy on HTTP tunnels detection. Furthermore, since it needs so few training samples, i.e. 100, the training time is quite short. This gives our proposed method another advantage of deployment. Currently we tested our proposed method offline. Online HTTP tunnel detection would be our further work.

References

[1] Julius Plenz, "DNS tunnel," http://www.dnstunnel.de/, Aug. 2010.

[2] Lars Brinkhoff, "GNU httptunnel," http://www.nocrew.org/software/httptunnel.html, Aug. 2010.

[3] Richard Mills, "The Linux Academy HTTP Tunnel," http://the-linux-academy.co.uk/downloads.htm, Aug. 2010.

[4] Internet Assigned Numbers Authority. http://www.iana.org/assignments/port-numbers. Aug. 2010.

[5] A. Madhukar and C. Williamson, "A longitudinal study of P2P traffic classification," 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 179-188. Sep. 2006.

[6] Lawrence Berkeley National Laboratory, "Bro Intrusion Detection System", http://www.bro-ids.org/, Aug. 2010.

[7] V. Paxson, "Bro: A system for detecting network intruders in real-time," Computer Networks, no. 31(23-24), pp. 2435-2463, Dec. 1999.

[8] Sourcefire, "Snort IDS/IPS," http://www.snort.org/.Aug. 2010.

[9] T. Choi, C. Kim, S. Yoon, J. Park, B. Lee,

H. Kim, and H. Chung, "Content-aware internet application traffic measurement and analysis," Proceedings of the 2004 IEEE/IFIP Network Operations and Management Symposium (NOMS'04), vol. 1, pp. 511-524, Apr. 2004.

[10] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," Proceedings of the 13th international conference on World Wide Web, pp. 512-521. May 2004.

[11] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "Acas: Automataed construction of applicatin signatures," SIGCOMM MineNet Workshop, pp. 107-202, Aug. 2005.

[12] A. Moore, and K. Papagiannaki, "Toward the accurate identification of network applications," Proceedings of 6th passive active measurement workshop (PAM), vol. 3431, pp. 41-54, Apr. 2005.

[13] V. Paxson, "Empirically derived analytic models of wide-area TCP connections," IEEE/ACM Transactions on Networking, vol. 2, no. 4, pp. 316-336, Aug. 1994.

[14] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," Proceedings of ACM/SIGCOMM Internet Measurement Conference (IMC) 2004, Taormina, Sicily, Italy, pp. 135-148, Oct. 2004.

[15] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semisupervised network traffic classification," ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS) Performance Evaluation Review, vol. 35, no. 1, pp. 369-370, Jun. 2007.

[16] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and Discriminating Between Web and Peer-to-Peer traffic in the Network Core," Proceedings of the 16th International World Wide Web Conference (WWW), Banff, Canada, pp. 883-892, May 2007.

[17] Samruay Kaoprakhon, and Vasaka Visoottiviseth, "Classification of audio and video traffic over HTTP protocol," Proceedings of the 9th international conference on Communications and information technologies, Incheon, Korea, pp. 1534-1539, Sep. 2009.

[18] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Detecting HTTP tunnels with statistical mechanisms," Proceedings of the 42nd IEEE International Conference on Communications (ICC 2007), Glasgow, Scotland, pp. 6162-6168, Jun. 2007.

[19] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," ACM SIGCOMM Computer Communication Review, vol. 37, issue 1, pp. 7-16, Jan. 2007.

[20] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting," Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 53, issue 1, pp. 81-97, Jan. 2009.

[21] Wireshark Foundation, "Wireshark", http://www.wireshark.org/, Aug. 2010.

[22] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," Data Mining and Knowledge Discovery, vol. 2, pp. 121-167, Jun. 1998.

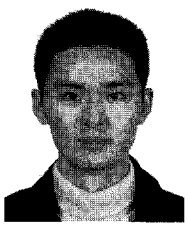
[23] C.C. Chang and C.J. Lin, "LIBSVM : a library for support vector machines," http://www.csie.ntu.edu.tw/~cjlin/libsvm/, Aug. 2010.

[24] Jon Postel, "Transmission Control Protocol," RFC793, Sep. 1981.

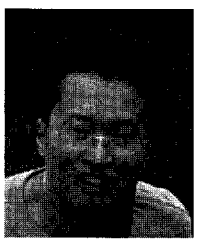
[25] Robert Russo, "Bayesian and Neural Net-

works for Motion Picture Recommendation," Ph.D thesis, Boston College. May 2006.
[26] I.H. Witten, and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 1st Ed., Morgan Kaufmann, San Francisco, CA, ISBN: 978155860-5527, Oct. 2005.
[27] f Irina Rish, "An empirical study of the naive Bayes classifier," IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, pp. 41-46, Jan. 2001.
[28] Riyad Alshammari, and A. Nur Zincir-Heywood, "Machine learning based encrypted traffic classification: identifying SSH and skype," Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications, Ottawa, Ontario, Canada, pp. 1-8, Jul. 2009.
[29] Yoav Freund, and Robert E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," Proceedings of the Second European Conference on Computational Learning Theory, pp. 23-37, Mar. 1995.

〈著者紹介〉



하 등 과 (DengKe He) 학생회원
2003년 6월: 해군잠수정대학교 정보관리학과 졸업
2006년 6월: 중경우전대학교 컴퓨터과학과 석사
2009년 3월~현재: 인하대학교 정보공학과 석사과정
〈관심분야〉 정보보호, 네트워크 보안



양 대 헌 (DaeHun Nyang) 종신회원
1994년 2월: 한국과학기술원 과학기술대학 전기·전자공학/컴퓨터공학과 졸업
1996년 2월: 연세대학교 컴퓨터과학과 석사
2000년 8월: 연세대학교 컴퓨터과학과 박사
2000년 9월~2003년 2월: 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재: 인하대학교 컴퓨터정보공학부 부교수
〈관심분야〉 암호 이론, 암호 프로토콜, 인증 프로토콜

이 경 희 (KyungHee Lee) 정회원
1989년: 서울대학교 식품영양학과 학사
1993년: 연세대학교 전산과학과 학사
1998년: 연세대학교 컴퓨터과학과 석사
2004년: 연세대학교 컴퓨터과학과 박사
1993년 1월~1996년 5월: LG소프트(주) 연구원
2000년 12월~2005년 2월: 한국전자통신연구원 선임연구원
2005년 3월~현재 : 수원대학교 조교수
〈관심분야〉 영상처리, 컴퓨터비젼, 인공지능, 패턴인식, 생체인식, 얼굴인식, 다중생체인식