A Novel Web Tunnel Detection Method Based on Protocol Behaviors

Fei Wang[*], Liusheng Huang, Zhili Chen, Haibo Miao, and Wei Yang

National High Performance Computing Center at Hefei,
Department of Computer Science and Technology,
University of Science and Technology of China,
Hefei, Anhui, 230027, P.R. China
wf616528291@gmail.com

Abstract. The web tunnel is a common attack technique in the Internet and it is very easy to be implemented but extremely difficult to be detected. In this paper, we propose a novel web tunnel detection method which focuses on protocol behaviors. By analyzing the interaction processes in web communications, we give a scientific definition to web sessions that are our detection objects. Under the help of the definition, we extract four first-order statistical features which are widely used in previous research of web sessions. Utilizing the packet lengths and inter-arrival times in the transport layer, we divide TCP packets into different classes and discover some statistical correlations of them in order to extract another three second-order statistical features of web sessions. Further, the seven features are regarded as a 7-dimentional feature vector. Exploiting the vector, we adopt a support vector machine classifier to distinguish tunnel sessions from legitimate web sessions. In the experiment, our method performs very well and the detection accuracies of HTTP tunnels and HTTPS tunnels are 82.5% and 91.8% respectively when the communication traffic is above 500 TCP packets.

Keywords: web tunnel detection, protocol behaviors, packet analysis, feature vector, support vector machine.

1 Introduction

In contemporary, people rely more and more on computers and the Internet. Network applications such as webpage browsing, business e-mail exchanging, microblog posting and online shopping become indispensable elements in people's jobs and daily lives. The accelerating rise in the demand for those techniques motivates diverse attacks from malicious users. The attackers utilize the legitimate-looking traffic generated by application-layer protocols (ALP) widely used in the Internet to launch imperceptible intrusions like implanting computer viruses, exposing sensitive information and filching confidential files.

At present, firewalls and application level gateways (ALG) configured to secure network boundaries can frustrate most bare attacks, for example downloading Trojans

[*] Corresponding author.

from offensive websites and access violations from the extranet to the intranet. In their existing incarnations, they usually protect local networks from damages by concentrating on controlling which websites local hosts are allowed to visit and which ALPs are permitted for communications. In order to achieve this goal, security policies of firewalls and ALGs are commonly implemented as filter criteria intercepting packets containing prohibited IP addresses or typical features of unapproved ALPs. In general, the two categories of devices operate cooperatively to enforce the criteria: the firewall checks IP addresses and port numbers while the ALG judges whether the traffic of a certain protocol conforms to the corresponding rules. For instance, in the case of a network only approving the non-encrypted Internet browsing, the firewall is the first defensive line which merely hands outgoing (ingoing) TCP packets, to (from) port 80 or 8080 at IP addresses not in the forbidden set, over to the ALG. The ALG then scrutinizes the format of the HTTP content in order to ensure that the peers are really "speaking" HTTP. Additionally, the ALG can also rule out potential malicious behaviors by denying particular strings in some vulnerable fields such as URLs, Hosts, User-Agents and values of different keys.

Upon most occasions, firewalls and ALGs can deal with bare intrusions with ease. However, some artful covered attacks have been devised in the past decade. Those techniques can successfully bypass the filter criteria utilized in firewalls and ALGs as legitimate applications and the application-layer tunnel (ALT) is a celebrated one among them. Nowadays, the ALT has become a threat which can't be overlooked to the Internet. The ALT is easy to be implemented but extremely difficult to be detected. The main idea of the ALT is disguising an ALP as another. The technique carries out the tunneling process by encapsulating the traffic of prohibited ALPs inside the payload of allowed ALPs. What's more, the formats of the embedded traffic can be various so that the ALT can perform obfuscation of original data to thwart pattern-matching classifiers based on the formats of ALPs employed by ALGs. The ALT can be implemented in two ways: one is in clear-text ALPs (CALT) and the other is in encrypted ALPs (EALT). As Fig. 1 shows, the CALT has a transport-layer shell and an unsuspicious header of a permitted clear-text ALP and the forbidden traffic is camouflaged in the body of entity data. Illustrated in Fig. 1, the EALT only has a transport-layer shell of an allowed encrypted ALP while the encrypted payload

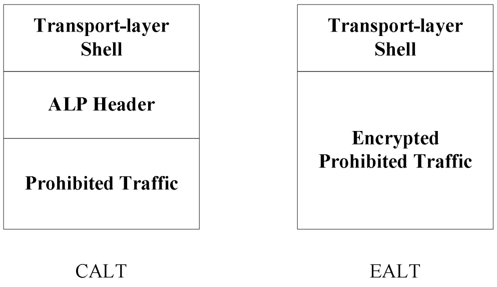

Fig. 1. The message structure of the two kinds of ALTs

is actually the encrypted prohibited traffic. In practice, because of the ubiquity of the web usage, the HTTP tunnel and the HTTPS tunnel are prevalent implementations for the CALT and the EALT respectively. As a result, the detection method we propose in this paper is aimed at the two ALT implementations.

1.1 Outline of Our Contributions

We propose a novel detection method to thwart web tunnels. Our technique is designed based on the protocol behaviors of HTTP and HTTPS. By analyzing the interaction processes in web communications, we give a scientific definition to web sessions which are our detection objects. Under the help of the definition, we firstly extract four first-order statistical features which are widely used in previous research of web sessions. Further, we concentrate on the TCP packets in the transport layer. Utilizing the packet lengths and inter-arrival times, we divide TCP packets into different classes and dig out the statistical correlations of them. With these correlations, we extract another three second-order statistical features of web sessions. Then, the seven features we have obtained can compose a 7-dimentional feature vector and we believe that the vector can fully reflect the statistical characteristics of web traffic. Exploiting the vector, we adopt a support vector machine classifier to distinguish tunnel sessions from legitimate web sessions. In our method, the detection rates of HTTP tunnels and HTTPS tunnels can reach above 80% and 90% respectively when the communication traffic is above 500 TCP packets.

1.2 Paper Organization

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 introduces some important notions and techniques used in this paper. Section 4 describes the work flow of our detection method. Section 5 discusses the process of the protocol feature extraction. Section 6 shows the detection results of our method. Section 7 is the conclusion.

2 Related Work

Early web tunnel detection is executed only at the application layer. Borders and Prakash proposed one of the first mechanisms to detect HTTP tunnels, named “Web Tap” [1]. The filter is designed to reveal covert communications tunneled in the HTTP traffic. The Web Tap depends on the simple analysis of features at the application layer, like HTTP transaction rates, transaction times, access frequency, etc. Strictly speaking, the analysis is coarse so that it may cause many false positives and false negatives, resulting in an unreliable system. Bissias et al. invented a statistical technique which can infer the source in HTTPS streams [2]. The technique shapes a website usually visited by a tuple with two elements: the size profile and the time profile. Given a website, the authors collect the HTTP packet sequences composing each HTTP request for the website. Then the mean value sequence of the lengths of these packets is the size profile and the mean value sequence of the inter-arrival times between the same packets is the time profile. HTTPS traces to be tested will be compared to the shapes of websites with the similarity computed from the cross-correlation between the sequences, making it possible to discover the destination. Subsequently, Liberatore et al. improved the technique and they extended the work to each URL to assign a given trace to a gathered profile, the Jaccard and the naïve Bayes similarity metrics [3]. Campos et al. exploited a clustering scheme to recognize different traffic patterns by a sequence of application-layer triples containing interactive features between peers: the length of data from the client, the length of data from the server and the time interval between data pairs [4].

Recently, researchers turned to the transport layer in order to devise advanced detection techniques and many methods based on both application-layer features and transport-layer features were worked out. McGregor et al. used many features of TCP packets to cluster the traffic so as to identify different ALPs [5]. The features include packet length, length percentile, etc. Moore and Zuev proposed a naive Bayes classifier to recognize ALPs and the results of classification were excellent [6]. The method firstly deploys some deep inspection in the TCP packets to dig out almost all the available features in TCP flows. Then it utilizes statistical approaches to kick out irrelevant factors in the feature vector, which can significantly improve the performance. Wright et al. adopted a similar technique to propose a k-nearest-neighbors classifier according to the K-L distance between feature vectors form different ALPs and the technique performed well in the identification of HTTP and HTTPS [7].

Most of the techniques listed above are utilized for traffic classification. They operate on the features collected from massive packets generated by ALPs and the experimental results are considerable. Nevertheless, the effects on the ALT of these classifiers have not been demonstrated. The ALT has the same transport-layer shell as that of normal ALPs, which may disable the classifiers. The detection method which is really effective on the HTTP tunnels was proposed by Crotti et al [8]. The technique runs on the fingerprint of ALPs and has a high detection rate against the HTTP tunnel. The fingerprint is trained from single TCP flows and it consists of two elements: TCP packet length sequence and TCP packet inter-arrival time sequence. Although the method can find out most HTTP tunnels, it still has some flaws in the fingerprint measurement which is its core concept. The packet classification in the fingerprint evaluation is coarse and the technique ignores many remarkable protocol behaviors in HTTP, so the fingerprint detection is inefficient and it is vulnerable in theory.

3 Preliminaries

In this section, we will introduce some important notions and useful techniques which can help us expose protocol behaviors in HTTP utilized in our detection method.

3.1 HTTP Flow and HTTP Session

In the fingerprint detection technique, the authors define an HTTP flow by a pair of reversed TCP flows: the client to the server and the server to the client. In computer networks, a TCP flow can be defined as a tetrad: source IP address, destination IP address, source port number and destination port number, while absolutely an HTTP flow can't be defined in the same way. When browsing a website, the requests for associated objects of an HTML file can be sent on other ports different from the original one which sends the request for the text webpage. Especially, in the keep-alive visiting method, requests for different HTML files may be sent on the same port. In addition, the HTTP server can also be a set of hosts. As we know, the associated objects of HTML files can be stored in different servers (it is extremely popular for large websites). As a result, the requests for an HTML file can refer to more than one server and we can't define the HTTP server as a single host. From the above, the definition of HTTP flows in the fingerprint detection technique is obviously unreasonable and it will doubtlessly lose many crucial characteristics of HTTP flows.

The scenario in Fig. 2 shows the amounts of HTTP requests at different times when a client is browsing a website. In this figure, we can see that the request distribution has crests and troughs. Each crest is a busy time interval with the outburst in requests and every trough is a silent period containing no requests. In terms of the behaviors of HTTP, a crest represents the requests for an integrated HTML file including the text webpage and its associated objects, while a trough between two consecutive crests is the time for visitors to handle the documents, such as reading news, thinking over a problem and saving elements. According to this, an HTTP flow can be defined as a pair of crests: a request crest in the client and the corresponding response crest in the server. The definition is not limited to a tetrad and it can scientifically describe the features in HTTP. The trough is called "Think Time" [4] and it is a significant factor in the analysis of HTTP flows. In practice, the length of the silent period is flexible and it differs from user to user. Therefore, the "Think Time" is not a main object in the study of HTTP, but it is only used to separate different HTTP flows. HTTP flows will be cut into pieces if the "Think Time" is too short. Considering the response time of the human body and the network delay, we hold the view that a reasonable "Think Time" should be longer than 5 seconds, which means that any two consecutive crests, with a trough less than or equal to 5 seconds between them, should belong to the same HTTP flow.

With the concepts discussed above, we can obtain the complete definition of HTTP flows. An HTTP flow consists of a series of requests from a client to a server and the corresponding responses from the server to the client. The interval between adjacent requests should not be more than 5 seconds. The client is a single host defined as an IP address. The server is a set of hosts with similar IP addresses. In general, the different servers in a large website are deployed in the same subnet. Hence, we can define the similar IP addresses as follows. If we have two IP addresses denoted by two 32-bit (IPV4) unsigned integers *ipa* and *ipb*, we will say they are similar when:

$$\begin{cases} ipa \oplus ipb \le 255 & \text{if } ipa \text{ and } ipb \text{ both belong to Class C} \\ ipa \oplus ipb \le 65535 & \text{if } ipa \text{ and } ipb \text{ both belong to Class B} \\ ipa \oplus ipb \le 16777215 & \text{if } ipa \text{ and } ipb \text{ both belong to Class A} \end{cases} \quad (1)$$

Where ⊕ is the operator of the binary "XOR". Owning the definition of HTTP flows, an HTTP session can be easily defined as a set of HTTP flows with the same client and server in a certain period (the interval between any adjacent flows in a session should be less than or equal to this period). In this paper, we set this period to half an hour. Since HTTPS is the encrypted version of HTTP and there is no essential discrepancy in them, HTTPS flows and sessions can be defined in the same way.

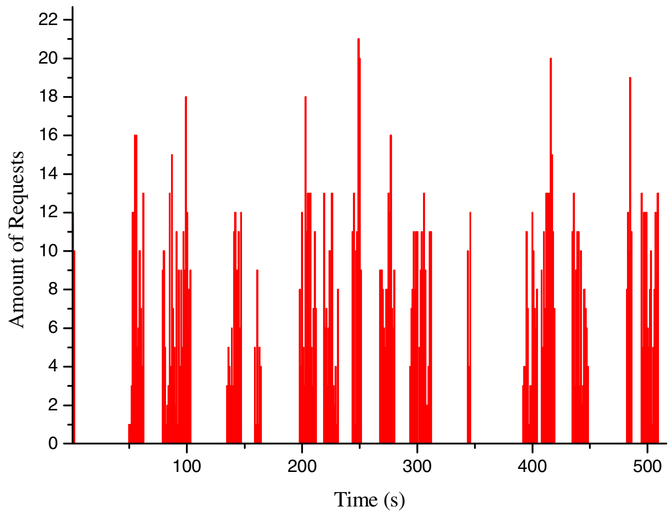

Fig. 2. The distribution of requests in the browsing

3.2 Kernel Density Estimation

The Kernel Density Estimation (KDE) is a practical method to approximate the distribution of a random variable [9]. Here, we can utilize the technique to estimate the probability density functions (PDF) of some crucial parameters in our detection method. For a random variable X, if we have n samples $\{x_1,x_2,\ldots,x_n\}$, then we can approximate the distribution of X by the KDE:

$$\hat{f}(x)=\frac{1}{nh}\sum_{i=1}^{n}K(\frac{x-x_i}{h}) \tag{2}$$

Where $\hat{f}(x)$ is the estimated PDF of X, h is called the kernel bandwidth, and $K(\bullet)$ is the kernel which can be any non-negative function satisfying:

$$\int_{-\infty}^{+\infty}K(g)\,dg=1 \tag{3}$$

In general, the kernel is assumed as a Gaussian distribution whose standard deviation is 1, because Gaussian distribution has desirable smoothness properties:

$$K(g) = \frac{1}{\sqrt{2\pi}} e^{\frac{-g^2}{2}} \tag{4}$$

Figure 3 [9] illustrates how an estimate of the PDF is constructed. For each data point, a Gaussian distribution centered on the very point is fitted. The summation of those functions gives an estimate of the real PDF.

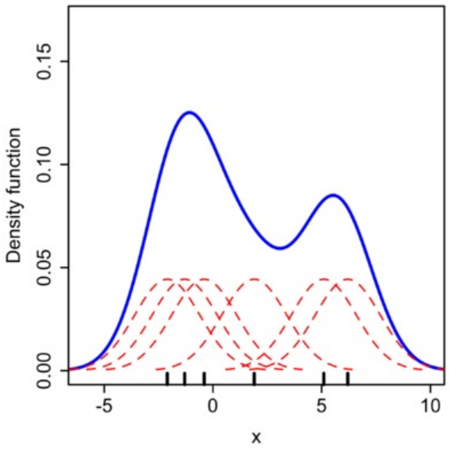

Fig. 3. The construction of an estimated PDF

The kernel bandwidth *h* plays an important role in the accuracy of the approximation. In practical estimation of the bandwidth, if the kernel is a Gaussian distribution, *h* can be optimized as [9]:

$$\hat{h} = (\frac{4\sigma^5}{3n})^{\frac{1}{5}} \approx 1.06\sigma n^{-\frac{1}{5}} \tag{5}$$

Where σ is the standard deviation of the samples.

4 Our Tunnel Detection Method

As mentioned before, our detection concentrates on the web tunnels (HTTP and HTTPS tunnels). If the length of a flow is short, the limited quantity of information in it will not sufficiently support us to judge whether it is a tunnel correctly. For this reason, our detection objects are HTTP and HTTPS sessions but not flows. Different from previous techniques, in our detection method, we discover some novel statistical features from protocol behaviors based on TCP packets in HTTP and HTTPS sessions. With these features, we can successfully identify the tunneled traffic in suspicious sessions. In the analysis of the TCP packets in sessions, we only focus on their lengths and inter-arrival times.

The work flow of our detection method is illustrated in Fig. 4. First, we collect plentiful legitimate sessions to obtain some significant statistical characteristics of

TCP packets in normal HTTP and HTTPS. Then, with the help of those characteristics, we extract seven features of suspicious sessions to be tested and those features are denoted by a 7-dimentional vector **FV**. Last, the vector **FV** is submitted to a classifier and the classifier identifies the types of the sessions. The classifier exploited here is a two-class Support Vector Machine (SVM) and we utilize the LIBSVM [10] as our tool. The SVM only identifies the suspicious sessions as two categories: normal HTTP (HTTPS) sessions and tunnel sessions, and the SVM is trained by feature vectors from the two categories of sessions in advance. We select the SVM as the classifier for our detection method due to its outstanding performance. Actually, if we collect tunnel sessions to compute the characteristics, the method can also operate well. But, after all, tunnel sessions are a tiny minority and to get massive normal HTTP and HTTPS sessions is much easier for us.

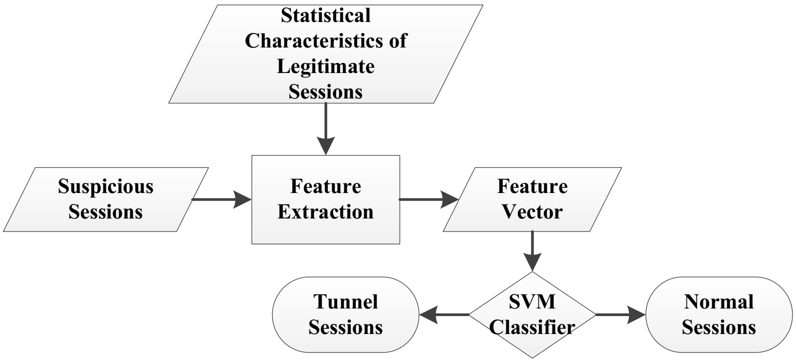

Fig. 4. The diagram of the work flow

5 Feature Extraction

In this section, we will discuss the seven features we collect from suspicious sessions. Four of the features are first-order statistics and the other three are second-order which can reveal the correlations between packets in sessions. The seven features are all extracted according to protocol behaviors. Hereafter, we use “HTTP” to represent both “HTTP” and “HTTPS”.

5.1 First-Order Features

As Section 3 explains, HTTP requests can be sent in different ports and each port can send more than one request. In the observation of HTTP flows, we can find out that in a single TCP connection the next request will not be sent until the response from the server to the former one arrives. Specifically, if we denote a request by *req* and a response by *res*, in a TCP connection, we can only get the sequence (*req*, *res*, *req*, *res*, *req*, *res*), but can’t get the sequence (*req*, *req*, *req*, *res*, *res*, *res*). Hence, we can easily calculate the sizes (Bytes) of requests and the sizes of their corresponding responses. Further, we can discover the following four first-order features in sessions.

Feature 1: Average Request Size (Req_{avg}). Req_{avg} is the arithmetic average of the sizes of HTTP requests in a session.

Feature 2: Request Size Variance (Req_{var}). Req_{var} is the variance of the sizes of HTTP requests in a session.

Feature 3: Average Response Size (Res_{avg}). Res_{avg} is the arithmetic average of the sizes of HTTP responses in a session.

Feature 4: Response Size Variance (Res_{var}). Res_{var} is the variance of the sizes of HTTP responses in a session.

The four simple features are widely used in previous work by other forms [1, 2, 3, 4, 5, 6]. The features are useful but obviously not enough because they don't have absolute exclusiveness in the detection.

5.2 Packet Classification

As what previous work did, in order to analyze TCP packets, we should divide them into different classes. We denote a TCP packet by a tuple with three elements: the packet length (exclude TCP header), the inter-arrival time and the direction. The direction here has two optional values: 0 (the packet is from the client to the server) and 1 (the packet is from the server to the client). The inter-arrival time represents the interval between the packet and the former one which has the same direction. We assume that there are no direct correlations between packets from different flows in a session, so the inter-arrival times are only computed in each flow respectively and the inter-arrival time of the first packet with either direction in each flow is regarded as 0. Flows are independent packet sequences in a session and flows are separated by the "Think Time". The TCP packets mentioned here do not contain the TCP control packets whose lengths are zero, such as SYN, FIN, RST and pure ACK, because they are irrelevant to HTTP behaviors. The coming problem is how to divide packet lengths and inter-arrival times into different bins. We utilize the KDE technique mentioned in Section 3. In the Ethernet, the maximum segment size (MSS) of TCP packet is 1460 bytes, so the packet lengths are between 1 and 1460. Owing to the "Think Time", the inter-arrival times are between 0 and 5000 (the unit is millisecond). The situation may happen that because of the network delay, the inter-arrival times of some packets from the server to the client in a flow may be a little longer than 5000ms, and we regard these times just as 5000ms. We select all the packets in the collected legitimate sessions as samples to estimate the PDF. We denote the counts of occurrences of the 1460 lengths and 5001 times by $\{CL_1, CL_2, \ldots, CL_{1460}\}$ and $\{CT_0, CT_2, \ldots, CT_{5000}\}$. Then, the PDF of the length $\widehat{f}(l)$ and the inter-arrival time $\widehat{f}(t)$ can be estimated by the KDE technique with the Gaussian kernel as the following formulas.

$$\widehat{f}(l) = \frac{1}{\sqrt{2\pi}\,\widehat{h}_l \sum_{j=1}^{1460} CL_j} \sum_{i=1}^{1460} CL_i \times e^{\frac{-(l-i)^2}{2\widehat{h}_l^2}} \qquad (6)$$

$$\hat{f}(t) = \frac{1}{\sqrt{2\pi}\hat{h}_t \sum_{j=0}^{5000} CT_j} \sum_{i=0}^{5000} CT_i \times e^{\frac{-(t-i)^2}{2\hat{h}_t^2}} \tag{7}$$

The outline of the packet length division is as follows and the inter-arrival time division can be done by the same method.

Step 1. Calculate cumulative probability *CP* between 1 and 1460:

$$CP = \int_1^{1460} \hat{f}(l)\, dl \tag{8}$$

Step 2. If we want to divide the lengths into *b* bins, we can split the interval [1, 1460] into *b* segments by *b*-1 cut points in ascending order $\{L_1, L_2, \ldots, L_{b-1}\}$. We can denote 1 and 1460 by L_0 and L_b, and then we can compute the cut points by:

$$\int_{L_i}^{L_{i+1}} \hat{f}(l)\, dl = \frac{CP}{b} \quad (0 \le i \le b-1) \tag{9}$$

This division scheme is closely related to the real distribution of the two elements, which will help improve the detection performance remarkably. If we have *BL* bins of the packet lengths and *BT* bins of the inter-arrival times, we can obtain $2 \times BL \times BT$ classes of TCP packets. In the KDE of packet length, we rule out the packets from the servers because the overwhelming majority of them are in the size of 1460, which can cause the classification useless. It should be noted that $\hat{f}(l)$ and $\hat{f}(t)$ are non-integrable functions, so we adopt the infinitesimal method to compute the integral and the infinitesimal is 0.05. Then, in the collected legitimate sessions, we can count the packets and compute the occurrence probability of each class by the maximum likelihood estimation. In the estimation, we adopt the Good-Turing smoothing technique to handle the packet classes which are not observed [11].

5.3 Second-Order Features

The packet distribution is a typical feature in sessions. By the packet classification, we can get the occurrence probabilities of the $2 \times BL \times BT$ packets in legitimate sessions. Then we can utilize the K-L divergence to measure the difference between the packet distribution in suspicious sessions and that of legitimate sessions.

Feature 5: Packet Distribution Difference (D_{KL}). D_{KL} is the K-L divergence of legitimate packet distribution from suspicious packet distribution:

$$D_{KL} = \sum_{i=1}^{2 \times BL \times BT} P(i) \ln \frac{P(i)}{Q(i)} \tag{10}$$

Where *P*(*i*) is the discrete probability density function of the packets in suspicious sessions and *Q*(*i*) is the function for the normal ones. In the computation, we needn't take special handling on the situation $0 \ln 0$ because we have the equation:

$$\lim_{x \to 0^+} x \ln x = 0 \tag{11}$$

So, we can just interpret $P(i)\ln\frac{P(i)}{Q(i)}$ as zero when $P(i)$ is zero ($1 \le i \le 2 \times BL \times BT$). Because we implement the Good-Turing smoothing technique in the maximum likelihood estimation, we can ensure that $Q(i) > 0$.

Flows in sessions can be expressed as different TCP packet sequences. HTTP has some particular interactive behaviors which can hardly be found in other ALPs. For example, when a packet from the server arrives at the client, the client will scan the data in the packet immediately. If some object links are in this packet or can be computed from the data in this packet, the client will send the requests on other ports dynamically regardless of whether the current response has been received integrally. In practice, these behaviors can be fully reflected by the packet sequences. So, we consider that the ordered packets in a flow are closely related and a packet is related to the ones nearby. In terms of protocol behaviors, correlations in packets are not merely restricted in the adjacent packets, but they are usually regarded as the collocations in some consecutive packets. In our detection method, we investigate pairs of packets within a certain range *N* (*N* is an integer greater than 1) in order to find out packet collocations in HTTP. For the analysis of collocations, we can adopt some techniques widely used in the statistical natural language processing [12].We define that any two packets in the same flow with a distance less than *N* is a packet pair and we denote a packet pair "*xy*" by an ordered pair <*x*,*y*>. These packet pairs are called N-Range Packet Pairs (N-RPP) [12]. Figure 5 illustrates an example for 3-RPPs, and in the dashed box we can observe three 3-RPPs: <2,5>, <2,4> and <5,4>. The numbers in packet pairs here are labels of packet classes.

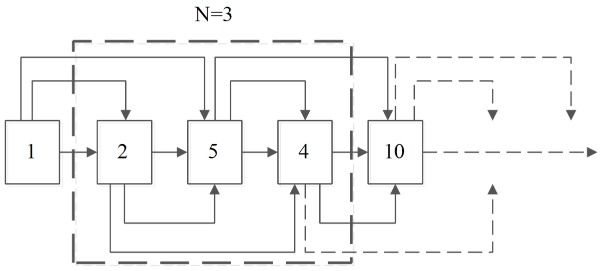

Fig. 5. An example for 3-RPPs

Because we have $2 \times BL \times BT$ packets, we can obtain $(2 \times BL \times BT)^2$ N-RPP theoretically. The occurrences of N-RPPs are prominent features in HTTP sessions. The entropy is usually utilized to evaluate the uncertainty and the potential regularity in random variables. For different protocols, the regularities of behaviors have significant difference, so we can also use the entropy to extract a feature of N-RPPs in sessions.

Feature 6: N-RPP Entropy ($E_{N\text{-}RPP}$). We regard the N-RPP as a random variable and $E_{N\text{-}RPP}$ is its entropy:

$$E_{N\text{-}RPP} = -\sum_{i=1}^{(2\times BL\times BT)^2} F(i)\log_2 F(i) \tag{12}$$

Where $F(i)$ is the occurrence probability of the ith N-RPP ($1 \le i \le (2\times BL\times BT)^2$). Similar to Equation (11), when $F(i)$ is zero, we can treat $F(i)\log_2 F(i)$ as zero.

In the information theory, another measurement for discovering interesting collocations is the mutual information (MI). For two discrete random variables X and Y, the MI can be defined as:

$$MI(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log_2 \frac{p(x,y)}{p(x)p(y)} \tag{13}$$

Where x and y represent all the possible values of X and Y. Here, $p(x,y)$, $p(x)$ and $p(y)$ are the occurrence probabilities of "xy", "x" and "y". However, the packets in a sequence are usually treated as different points. So, we utilize the pointwise mutual information (PMI) which is more suitable than the MI to evaluate the correlations of packets. The PMI between two particular points x and y can be defined as:

$$PMI(x;y) = \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{14}$$

Here, in our case, we regard $p(x,y)$, $p(x)$ and $p(y)$ as the occurrence probabilities of the packet pairs "xy", "x?" and "?y", where the symbol "?" represents any packet.

Now we can obtain the definition of the N-Range Mutual Information (N-RMI). The N-RMI can be defined as the PMI of an N-RPP [12]. The N-RMI can measure the collocation degree of an N-RPP and the larger the N-RMI is, the more reasonable the occurrence of the corresponding N-RPP is. With the definition of the N-RMI, we can use Equation (14) to evaluate the N-RMI of a certain N-RPP <x,y>. Given an HTTP flow, we denote the counts of occurrences of any N-RPP, the N-RPPs <x,y>, <x,?> and <?,y> by C_{tot}, C_{xy}, $C_{x?}$ and $C_{?y}$ severally. Then, the N-RMI of the N-RPP <x,y> can be calculated by [12]:

$$N\text{-}RMI_{<x,y>} = \log_2 \frac{p(x,y)}{p(x)p(y)} = \log_2 \frac{C_{xy}/C_{tot}}{(C_{x?}/C_{tot})(C_{?y}/C_{tot})} = \log_2 \frac{C_{xy}C_{tot}}{C_{x?}C_{?y}} \tag{15}$$

Here, we give an example to explain the computation of the N-RMI further. Given a packet sequence "2,5,1,3,4,15,103,19,2,3,3", we can evaluate the 4-RMI of the 4-RPP <2,3>. All the 4-RPPs in this flow are: <2,5>, <2,1>, <2,3>, <5,1>, <5,3>, <5,4>, <1,3>, <1,4>, <1,15>, <3,4>, <3,15>, <3,103>, <4,15>, <4,103>, <4,19>, <15,103>, <15,19>, <15,2>, <103,19>, <103,2>, <103,3>, <19,2>, <19,3>, <19,3>, <2,3>, <2,3>, <3,3>. Then, C_{23}=3, $C_{2?}$=5, $C_{?3}$=8 and C_{tot}=27, so we have:

$$4\text{-}RMI_{<2,3>} = \log_2 \frac{C_{tot}C_{23}}{C_{2?}C_{?3}} = \log_2 \frac{3\times 27}{5\times 8} = 1.0179 \tag{16}$$

If an N-RPP doesn't emerge in the legitimate sessions observed, we can infer that the appearance of the N-RPP is impossible and its occurrence is extremely anomalous. Therefore, instead of executing the smoothing technique in the computation of the packet distribution, we give a custom infinitesimal value INFS to the N-RPPs not observed as their N-RMIs. We can utilize the N-RMI to extract another feature of HTTP sessions.

Feature 7: N-RMI Distance ($D_{N\text{-}RMI}$). In a session, we may have many different N-RPPs. We select M N-RPPs which have the first M greatest N-RMIs in a suspicious session because we believe these N-RPPs can well present statistical correlations of this session. If a session only has S N-RPPs, where $S < M$, we'll regard the N-RPP with the smallest N-RMI in this session as the surplus $M - S$ N-RPPs. Then, $D_{N\text{-}RMI}$ can be defined as [12]:

$$D_{N\text{-}RMI} = \sum_{i=1}^{2\times BL\times BT} \sum_{j=1}^{2\times BL\times BT} | N\text{-}RMI_{<i,j>} - \overline{N\text{-}RMI_{<i,j>}} | \times \sigma(i,j) \qquad (17)$$

Where $N\text{-}RMI_{<i,j>}$ is the N-RMI of the N-RPP $<i,j>$ in the suspicious session and $\overline{N\text{-}RMI_{<i,j>}}$ is the N-RMI in legitimate sessions. The expression $\sigma(i,j)$ is evaluated as:

$$\sigma(i,j) = \begin{cases} 1 & \text{if } <i,j> \text{ is one of the } M \text{ selected N-RPPs} \\ 0 & \text{else} \end{cases} \qquad (18)$$

6 Experiment

6.1 Data Collection

The data collection is a crucial factor for our detection results, so it will be discussed with detail in this section. We deploy our packet sniffer on the gateway of our department which owns about 350 personal computers to collect legitimate HTTP and HTTPS sessions. Within a month, we collect 14462 HTTP and 9154 HTTPS legitimate sessions. Additionally, we generate some tunnel sessions: 200 HTTP (HTTPS) sessions with FTP encapsulated, 200 HTTP (HTTPS) sessions with SMTP encapsulated and 200 HTTP (HTTPS) sessions with POP3 encapsulated. The usage of those sessions is listed in Table 1, where "PD&N-RMI" is the amount of sessions used to compute the packet distribution and N-RMIs in legitimate sessions, "Train" is the amount of session samples used to train our classifier and "Test" is the amount of session samples to be tested.

In an SVM classifier, if the data sets used for training the classification model are not balanced, the classification accuracy may not be a good criterion for evaluating the effect of classifying, which will invalidate the detection results in our experiments. Therefore, in order to balance the scales of data sets for higher detection reliability, we set the amount of legitimate HTTP (HTTPS) sessions for training to 300, which is just the sum of all the tunnel sessions for training.

Table 1. The usage of collceted sessions

	PD&N-RMI	Train	Test
HTTP	14062	300	100
HTTPS	8754	300	100
FTP over HTTP (HTTPS)		100	100
SMTP over HTTP (HTTPS)		100	100
POP3 over HTTP (HTTPS)		100	100

In the data collection, we should bring out some processing details. The inter-arrival times may be distorted by network jitters. So, in order to reduce the impact of network noises as far as possible, all the data are collected in a certain time period on weekdays (14:00-17:30) and all the tunnel sessions are also generated in the same period. To simulate normal communication scenarios, when generating tunnel sessions, we deploy one host in our department LAN and the other in the WAN outside. We utilize the HTTPTunnel [13] to generate HTTP tunnel sessions and the Barracuda HTTPS Tunnel [14] to generate HTTPS tunnel sessions. There are two modes of the tunnel traffic generation, with a proxy (intermediary forwarding) and without proxies (direct connection). Since the proxy can visibly slow down the protocol interaction, which can be easily caught, we utilize the mode for direct connection. As explained in Section 4, we can't judge a session with few packets, so the sessions used for training and testing in our detection method all have more than 500 packets. In Section 6.2, we will use some or all of those packets for experiments.

6.2 Results

To implement our detection method, we should set the parameters firstly. In the packet classification, *BL* and *BT* can't be too large, which may ruin the similarities of packets. So, in our detection method, *BL* is 20 and *BT* is 15. We set *N* and *M* to 3 and 25 respectively, so we focus on the 3-RPP and the 3-RMI. The custom infinitesimal INFS we utilize to evaluate the 3-RMIs of unobserved 3-RPPs is set to -50. Figure 6 shows the results of the KDE and we can see that the estimated PDF can approximate the real density well. In the experiments, we use both our detection method and the fingerprint detection technique to detect the tunnel sessions to make a comparison. In our detection, in both training and testing, each session is denoted by its feature vector **FV** consisting of Req_{avg}, Req_{var}, Res_{avg}, Res_{var}, D_{KL}, $E_{3\text{-}RPP}$, $D_{3\text{-}RMI}$. In the fingerprint detection, the fingerprint is trained from all the legitimate sessions collected by us. The results for HTTP and HTTPS tunnel detection are shown in Table 2 and Table 3, where "Packet" is the amount of packets in training and testing session samples, "Legitimate" is the amount of sessions identified as the legitimate sessions, "Tunnel" is the amount of sessions identified as the tunnel sessions, "HTTP (HTTPS)" represents the legitimate HTTP (HTTPS) sessions, "FTP" represents HTTP (HTTPS) sessions tunneled with FTP, "SMTP" represents sessions tunneled with SMTP and "POP3" represents sessions tunneled with POP3.

Table 2. Detection results of HTTP tunnels

Packet	Type	Our Detection Method			Fingerprint Detection Technique		
		Legitimate	Tunnel	Accuracy	Legitimate	Tunnel	Accuracy
100	HTTP	62	38	62%	44	56	44%
	FTP	31	69	69%	42	58	58%
	SMTP	37	63	63%	33	67	67%
	POP3	46	54	54%	48	52	52%
				62.0%			55.3%
300	HTTP	75	25	75%	65	35	65%
	FTP	22	78	78%	38	62	62%
	SMTP	29	71	71%	39	61	61%
	POP3	39	61	61%	43	57	57%
				71.3%			61.3%
500 or more	HTTP	81	19	81%	63	37	63%
	FTP	8	92	92%	29	71	71%
	SMTP	17	83	83%	45	55	55%
	POP3	26	74	74%	39	61	61%
				82.5%			62.5%

Table 3. Detection results of HTTPS tunnels

Packet	Type	Our Detection Method			Fingerprint Detection Technique		
		Legitimate	Tunnel	Accuracy	Legitimate	Tunnel	Accuracy
100	HTTPS	69	31	69%	52	48	52%
	FTP	28	72	72%	40	60	60%
	SMTP	26	74	74%	32	68	68%
	POP3	45	55	55%	47	53	53%
				67.5%			58.3%
300	HTTPS	73	27	73%	70	30	70%
	FTP	15	85	85%	35	65	65%
	SMTP	25	75	75%	36	64	64%
	POP3	20	80	80%	49	51	51%
				78.3%			62.5%
500 or more	HTTPS	87	13	87%	57	43	57%
	FTP	2	98	98%	32	68	68%
	SMTP	7	93	93%	21	79	79%
	POP3	11	89	89%	42	58	58%
				91.8%			65.5%

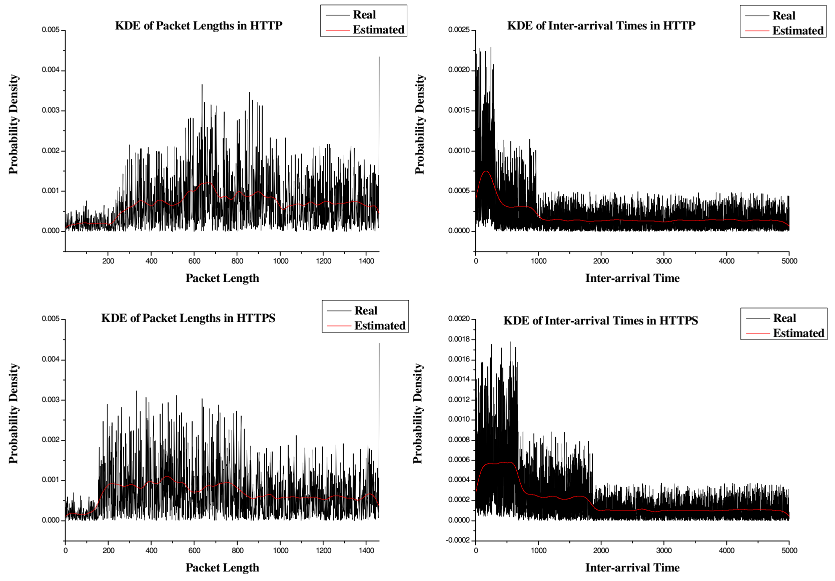

Fig. 6. Results of the KDE: up-left is the KDE of packet lengths in HTTP sessions, up-right is the KDE of inter-arrival times in HTTP sessions, down-left is the KDE of packet lengths in HTTPS sessions and down-right is the KDE of inter-arrival times in HTTPS sessions

As the results show, with the data collected, our detection method performs obviously better than the fingerprint detection technique does. Because we concentrate on protocol behaviors to extract useful features, our detection method is more effective. To improve the detection accuracy, the fingerprint detection technique needs many more sessions to train the fingerprint. With the amount of packets in sessions increasing, the detection rate rises significantly. Additionally, we can see that in our detection method the detection rate against HTTPS tunnels is higher. So, we can infer that protocol behaviors in HTTPS are more distinctive and pronounced than that in HTTP. Further, Table 4 and Table 5 show the detection results when we kick out partial features from the feature vector (the amount of packets in sessions is above 500). We can see that detection accuracies decrease significantly when we abandon the first-order features or the second-order features. Hence, we can conclude that the two categories of features both play important roles in our detection method.

Table 4. Results without some features in HTTP tunnel detection

Type	Without First-Order Features			Without Second-Order Features		
	Legitimate	Tunnel	Accuracy	Legitimate	Tunnel	Accuracy
HTTP	60	40	60%	53	47	53%
FTP	31	69	69%	30	70	70%
SMTP	53	47	47%	42	58	58%
POP3	46	54	54%	59	41	41%
			57.5%			55.5%

Table 5. Results without some features in HTTPS tunnel detection

Type	Without First-Order Features			Without Second-Order Features		
	Legitimate	Tunnel	Accuracy	Legitimate	Tunnel	Accuracy
HTTPS	72	28	72%	49	51	49%
FTP	21	79	79%	22	78	78%
SMTP	45	55	55%	48	52	52%
POP3	39	61	61%	33	67	67%
			66.8%			61.5%

7 Conclusion

In this paper, we devise a novel web tunnel detection method based on protocol behaviors. We extract seven useful statistical features according to the communication characteristics in HTTP and HTTPS. With those features, we utilize a SVM classifier to distinguish legitimate sessions and tunnel sessions. In the experiment, the detection accuracy of our method is much higher than that of the technique proposed in previous work.

In the future, we can research protocol behaviors in other applications. Obtaining the behavior characteristics, we can extend our feature extraction method to do further classification of network traffic.

Acknowledgement. This paper was supported by the National Natural Science Foundation of China (Nos.60903217 & 61202407), the Fundamental Research Funds for the Central Universities (Nos.WK0110000027 & WK0110000033), the Natural Science Foundation of Jiangsu Province of China (No.BK2011357), the Guangdong Province Strategic Cooperation Project with the Chinese Academy of Sciences (No.2012B090400013), and the Scientific and Technical Plan of Suzhou (No. SYG201010).

References

1. Borders, K., Prakash, A.: Web Tap: Detecting Covert Web Traffic. In: Proceedings of the 11th ACM Conference on Computer and Communication Security, pp. 110–120 (October 2004)
2. Bissias, G.D., Liberatore, M., Jensen, D., Levine, B.N.: Privacy Vulnerabilities in Encrypted HTTP Streams. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 1–11. Springer, Heidelberg (2006)
3. Liberatore, M., Levine, B.N.: Inferring the source of encrypted http connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, pp. 255–263 (2006)
4. Hernández-Campos, F., Smith, F.D., Jeffay, K., Nobel, A.B.: Statistical Clustering of Internet Communications Patterns. Computing Science and Statistics 35 (2003)

5. McGregor, A., Hall, M., Lorier, P., Brunskill, J.: Flow Clustering Using Machine Learning Techniques. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 205–214. Springer, Heidelberg (2004)
6. Moore, A.W., Zuev, D.: Internet traffic classification using bayesian analysis techniques. In: SIGMETRICS 2005: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Banff, Alberta, Canada, pp. 50–60 (2005)
7. Wright, C.V., Monrose, F., Masson, G.M.: On Inferring Application Protocol Behaviors in Encrypted Network Traffic. Journal of Machine Learning Research 7, 2745–2769 (2006)
8. Dusi, M., Crotti, M., Gringoli, F., Salgarelli, L.: Detecting Application-Layer Tunnels with Statistical Fingerprinting. Journal of Computer Networks 53(1), 81–97 (2009)
9. Wiki: Kernel Density Estimation (2013), `http://en.wikipedia.org/wiki/Kernel_density_estimation`
10. Chang, C., Lin, C.: LIBSVM: a library for support vector machines (2013), `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
11. Chen, S., Goodman, J.: An empirical study of smoothing techniques for language modeling. In: Proceedings of the 34th Annual Meeting on Association for Computational Linguistics (ACL 1996), NJ, USA, pp. 310–318 (June 1996)
12. Chen, Z., Huang, L., Yu, Z., Yang, W., Li, L., Zheng, X., Zhao, X.: Linguistic Steganography Detection Using Statistical Characteristics of Correlations between Words. In: Solanki, K., Sullivan, K., Madhow, U. (eds.) IH 2008. LNCS, vol. 5284, pp. 224–235. Springer, Heidelberg (2008)
13. HTTPTunnel v1.2.1 (2013), `http://sourceforge.net/projects/http-tunnel/files/http-tunnel/HTTPTunnel%20v1.2.1`
14. Barracuda HTTPS Tunnel (2013), `http://barracudadrive.com/HttpsTunnel.lsp`