

# Identifying the traffic of SSH-encrypted applications

Maurizio Dusi, Alice Este, Francesco Gringoli, Luca Salgarelli  
DEA, Università degli Studi di Brescia,  
via Branze, 38, 25123 Brescia, Italy  
E-mail: <firstname.lastname>@ing.unibs.it

**Abstract**—With cryptographic tunnels such as the ones provided by Secure Shell (SSH) the users protect their privacy on the Internet from two points of view. First, they aim at preserving the privacy of their data. Second, they expect that the information related to the operations they execute, e.g., the type of applications they use, also remains private. In this paper we analyse the statistical information that observing encrypted traffic can be used to break the second type of protection. We also report on two statistical traffic identification techniques applying them to SSH tunnels, at least under some restricting hypothesis. Experimental results show how current implementations of SSH can be susceptible to this type of analysis, and illustrate the effectiveness of our two classifiers both in terms of their capabilities in analyzing encrypted traffic and in terms of their relative computational complexity.

Session subject: Networks

## I. INTRODUCTION

Internet users looking for privacy usually resort to the application of cryptographic mechanisms to protect their activities. The type of protection that is normally expected covers both *the content* of the data as well as *the type of application* that is exchanging it. The second type of protection is particularly important because it relates to the ability of the users to keep the way they use the Internet private, besides of course keeping their data confidential. Exposing the applications that users employ to exchange data can lead to further loss of privacy, and possibly to the application of policies that might be detrimental to each user’s freedom, for example enabling the profiling of their activities. A very popular mechanism that is supposed to provide both types of protection is offered by the Secure Shell (SSH) protocol [1], which can securely tunnel any TCP-based application by means of strong cryptography.

In this paper we show that regardless of the obfuscation performed by the SSH protocol, one can successfully gain information about the application that is forwarded inside the tunnel by analyzing basic IP-level information such as the length of the packets composing the encrypted stream. We report on these issues by introducing three research contributions.

First, we prove that under relatively reasonable assumptions, e.g., a single application injects traffic on an SSH tunnel at any given moment, the behavior of users can be exposed even when their traffic is protected by cryptographic means by the simple exploitation of well known statistical analysis techniques, at least with the current SSH specifications. We analyze the amount of information useful to identify the application protocol injecting traffic on an SSH tunnel. We focus on the decreasing of the information quantities that

we can extract from the statistical features after the SSH-encryption. We show that this decrement is low and that the SSH-encryption does not inhibit us from discriminating the correct application protocol.

Second, we provide insights on the behavior of two of the most popular techniques at the basis of many traffic analysis mechanisms, i.e., Gaussian Mixture Models (GMM) and Support Vector Machines (SVM), when used to characterize SSH-encrypted sessions: in this context we study their relative effectiveness, precision and computational complexity.

Finally we describe the design and implementation of a software tool called “SSHgate” that can be used to collect SSH-encrypted sessions along with the corresponding clear-text traffic. The tool, besides being at the base of the datasets that were used to derive the results presented here, has been published as an open source package [2], and can therefore be useful to any research project that deals with SSH traffic.

The rest of the paper is organized as follows. In Section II we report about related work. In Section IV we describe the information measures we apply and we explain how we designed the two classifiers based on GMM and SVM. We show how to apply the two statistical classifiers to the recognition of SSH tunnels in Section V. Section VI describes the dataset we used to obtain the experimental results, which are in turn presented in Section VII. Finally, Section VIII concludes the paper.

## II. RELATED WORK

The attempt to apply statistical (behavioral) traffic classification techniques to the characterization of encrypted sessions is relatively recent. Wright et al. [3] showed that an encrypted IPSec tunnel carrying traffic sessions generated by the same application, leaks enough information about these sessions to allow to precisely assess their number. Moreover, in our previous work [4] we described a statistical classification technique able to detect when an SSH session is used to tunnel other protocols rather than used for remote administration or remote copy.

In Bernaille et al. [5], the information about packet size and packet direction is exploited by a clustering technique for classifying (i.e., assigning each session to a specific application class) encrypted traffic. Wright et al. in [6] describe a technique for the classification of encrypted traffic based on a Hidden Markov Model (HMM). The authors evaluate the technique on clear-text traffic that was previously encrypted in an “artificial way”: in fact, the size of each packet is rounded up to the next multiple of the block-cipher size. In our

previous work [7] we reported on a preliminary SSH channel model that can be used to help inferring the application layer protocol that is being encrypted over an SSH tunnel. The technique was evaluated on clear-text sessions previously captured from real links and “artificially” encrypted through the model, mimicking Wright et al.’s approach. The resulting sessions were then classified with a GMM-based technique.

In this paper, we move from the concept of SSH channel model and we consider traffic that was collected over actual SSH channels. Furthermore, we introduce another classifier, based on SVMs, designed to expose the application behind SSH tunnels, and we present a comparison between the two approaches. We also produce an analysis of the amount of the information that the sessions carry useful to recognize the applications behind SSH tunnels, using the information measures we show in the following Section.

### III. DESCRIPTION OF THE INFORMATION MEASURES

We use information measures to evaluate the capability of discriminating the application protocols injecting traffic on an SSH tunnel. We apply statistical measurements of the amount of information carried by the features we extract from the sessions generated by different applications. Let  $x$  be a statistical feature, of which we extract the value from each session, such as the size of the  $i$ -th packet or the number of packets composing the session. For each session we also indicate with  $y$  the index of the application protocol that generate the corresponding packet sequence. All the sessions generated by the same protocol have the same value of  $y$ .

We use *entropy* [8] as a measure of the average uncertainty associated with the value of a feature; it represents the average number of bits necessary to describe its value.

We also use *mutual information* measure  $I(x; y)$  between the extracted feature  $x$  and the feature denoting the protocol index  $y$ . The mutual information is a symmetric quantity in  $x$  and  $y$ , i.e.,  $I(x; y) = I(y; x)$ , and it measures the information contribution that each feature provides regarding the value of the other.  $I(x; y)$  is also a measure of the dependence between the values of the two features, it is a non-negative quantity and it is zero only when  $x$  and  $y$  are independent. We can also replace  $x$  with a vector of features, for evaluating the contribution of a set of features in discriminating the application protocol  $y$ .

To obtain the value of the mutual information we need to know an estimation of the densities of the features. Naturally, such densities are not known a-priori, especially since we are dealing with features of traffic sessions. Given a class of sessions, generated by the same application protocol, we approximate the feature distributions starting from a finite set of  $n$  samples corresponding to  $n$  sessions belonging to the class. We do not assume that the generation of the samples is ruled by a particular functional expression, therefore we adopt a non-parametric approach to density estimation [9], that defines the density in a point as the sum of kernel functions centered in the  $n$  samples. The kernel  $K_i : \mathbb{R}^n \rightarrow \mathbb{R}$  determines the density smoothing; we choose a multivariate

Gaussian kernel function. We obtain the density estimations, using the Expectation Maximization (EM) algorithm [10], [11], maximizing the likelihood function [9] on a different set of observations of the same class.

With an experimental validation we ensure that the descriptions of the protocols we achieve with this procedure are accurate according to the numerosness of samples we extracted from the traces.

### IV. DESCRIPTION OF THE GMM AND SVM TECHNIQUES

In this paper we apply two well-known supervised statistical techniques to the classification of SSH-encrypted traffic: Gaussian Mixture Model (GMM) and Support Vector Machine (SVM) [12]. The reason behind the choice of these techniques is twofold. On one hand, the SVM approach has shown to perform well when applied to the problem of clear-text traffic classification [13], [14], [15]. On the other hand, we preliminary investigated the ability of GMM to classify “artificially” encrypted traffic in our previous work [7].

In this work we evaluate the effectiveness of GMM with actual SSH traffic, i.e., real tunneled traffic collected over encrypted SSH channels. At the same time, we compare its effectiveness, computational complexity and precision with an SVM-based approach when applied to the characterization of SSH traffic.

These two methods belong to the category of the supervised classification techniques. Supervised approaches require two phases: a training and an evaluation phase. During the training phase, the algorithms “acquire knowledge” about the classes that we want to identify. During the evaluation phase, a classification mechanism associates an unknown observation to one of the available classes. We describe in the following how the algorithms are trained and how they can be used to assign unknown observations to the available classes.

#### A. Gaussian Mixture Model

We express the GMM [16] parametric distribution as

$$f(\mathbf{x}) = \sum_{i=1}^L a_i \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_i, \Sigma_i),$$

where  $L$  is the number of mixture components,  $a_i \geq 0$  are the mixing proportions ( $\sum_{i=1}^L a_i = 1$ ) and the set  $\{a_i, \boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^L$  represents all distribution parameters. Each Gaussian component  $\mathcal{N}$  is parametrized by a mean vector  $\boldsymbol{\mu}_i$  and a symmetric semi-positive definite covariance matrix  $\Sigma_i$ .

Given a set of training observations  $T_S = (\mathbf{x}_1, \dots, \mathbf{x}_n)$  belonging to the same class, the distribution parameters are computed using the Expectation Maximization (EM) algorithm. This iterative method estimates the parameters of the parametric distribution by maximizing the likelihood function.

To compute the GMM models for each training class, we have split into two parts the training samples to obtain a second set of observations for the optimization of parameters, according to the procedure we have described in our previous work [7]. To assign an unknown observation  $\mathbf{x}$  to one of the

$M$  available classes, we use the following threshold-based rejection schema:

$$\begin{cases} \arg \max_{i: f_i(\mathbf{x}) > T_i} \{f_i(\mathbf{x})\}, & \text{if } \exists i : f_i(\mathbf{x}) > T_i \\ \text{unknown} & \text{otherwise} \end{cases} \quad (1)$$

if the condition in the first row above is satisfied the candidate class is found. The threshold value  $T_i$  for each class is computed during the training phase: there we choose the threshold values that maximize the sum of the *F-measure* [17] metric over all the classes. Given a class, its *F-measure* metric is defined as:

$$2 \cdot \textit{precision} \cdot \textit{recall} / (\textit{precision} + \textit{recall}) \quad (2)$$

where *precision* is the ratio of true-positives (TP) of the class over the sum of TP and false-positives (FP), while *recall* is the ratio of TP over the sum of TP and false-negatives (FN). In our context, TP are the observations correctly assigned to the class that generated them. FP are the observations of other classes incorrectly assigned to the class, while FN are the samples incorrectly assigned to another class.

### B. Support Vector Machine

The second classifier is based on the Single-class SVM proposed by Schölkopf in [18]. Choosing a Gaussian kernel function, the SVM model for a given class is represented by the following sum:

$$f(\mathbf{x}) = \sum_{i=1}^L a_i \mathcal{N}(\mathbf{x} | \mathbf{y}_{SV_i}, \sigma), \quad (3)$$

where  $L$  is the number of Gaussians, each one centered at a *Support Vector*  $\mathbf{y}_{SV_i}$ , while  $\sigma$  is the standard deviation value, that is the same for all directions and for all Gaussians of the class. The *Support Vectors* are observations selected among the set of training samples of the corresponding class during the training phase.

The classifier trains a SVM model for each training protocol, determining in the feature space the region that contains the training observations with a reliability level depending on the parameters of the model. In this case there are three parameters: the standard deviation  $\sigma$  of the Gaussian functions, the regularization coefficient  $\nu$  and the threshold value  $\rho$ . The parameter  $\sigma$  is the same appearing in the Equation 3, while  $\nu$  is a coefficient that determines for each class the number of *Support Vectors* and the fraction of training errors, i.e., the observations of the training class that are not correctly identified. Lastly, the threshold value  $\rho$  allows to detect the samples belonging to other classes.

For selecting the values of these three parameters that allow to correctly identify the samples of the trained class and to exclude the elements of the other classes, we maximize the same *F-measure* metric defined by Equation 2, considering also the percentages of samples of the other training classes falling in the surface of the selected protocol (false positives).

The classification algorithm assigns a sample  $\mathbf{x}$  to the class whose decision function evaluated in  $\mathbf{x}$  takes the greatest

value. As in the GMM case, during the evaluation stage we consider only the class whose function values are larger than the corresponding threshold  $\rho_i$ . If all the values of the  $f_i(\mathbf{x})$  are lower than the thresholds, the observation  $\mathbf{x}$  is labeled as *unknown*, i.e., it does not belong to any of the training protocols.

## V. CLASSIFICATION OF SSH TUNNELS: FEATURE SELECTION

*Port forwarding* is one of the key features of the SSH protocol [1]: it allows users to forward any kind of network traffic running on top of TCP inside an SSH session thus providing confidentiality and integrity even to clear-text applications. Though encryption should guarantee that no information can be gained on the traffic forwarded via SSH, we explain how one could use the GMM and SVM-based classifiers to infer information about the tunneled application protocol. Before describing the experiment, we first discuss the port forwarding mechanism and we show how it affects simple IP-level features such as the length of the forwarded packets as they cross the SSH tunnel border. We then illustrate how from the observation of these features we can gather significant patterns to train the proposed statistical classifiers.

### A. The SSH port forwarding mechanism

The SSH protocol can, within the same client/server session, support multiple cryptographic “channels” which can be used to tunnel clear-text traffic (port forwarding), access remote servers through terminal emulation, etc. To request port forwarding users have to specify (i) a port number  $N$  that will be allocated by the SSH client process at the local side and (ii) a destination address that the remote SSH server will contact at (iii) the remote port number  $M$ . They can then configure a local application to connect to the local TCP port  $N$  instead of directly contacting the destination address at port  $M$ : the local SSH peer that holds the contacted socket asks its remote peer to connect to the final destination. The two SSH peers begin then to tunnel the application data inside the SSH session. From the application perspective there is no way to know that the exchanged data is crossing a tunnel. The resulting picture comprises three TCP sessions: two “outer” clear-text sessions, one from the original client application to the SSH client, one from the SSH server to the original application server, and the “inner” SSH-encrypted session.

Note that a single SSH connection can tunnel several TCP sessions at the same time: in this case, SSH assigns to each flow a separated “channel”, each with a specific SSH identifier. Since the encryption process hides the correspondence between channels and each tunneled application, the classification mechanisms described in this paper could not operate in presence of multiple tunnels supported at the same time by a single SSH connection. For this reason, in this paper we assume that each user will use an SSH connection to tunnel one application at any given time. Users could still tunnel multiple applications over the same SSH connection, but we assume they will do it one application at a time, without

multiplexing several applications at once, or multiplexing port forwarding and remote terminal or secure file copy.

Although this assumption restricts the applicability of our techniques, we believe it is reasonable to think that a significant fraction of SSH users today do use a given SSH connection to perform one activity at a time. We will study how to remove this assumption in a future work.

### B. Feature selection

Thanks to the hypothesis introduced at the end of the previous section we can now assume that each observed SSH packet is transporting the data generated by a single user application<sup>1</sup>.

The way SSH segments are sent through the SSH channel one after the other strictly depends on the length of the clear-text packets coming from the original port-forwarded application traffic which, in turn, depends on the finite state machine of the application protocol that generated them. Furthermore, the direction of the packets is preserved too: packets coming from the SSH client and going to the SSH server are the effect of the TCP segments that the outer client sent to the local TCP port, i.e., the port allocated on the SSH client to receive the application traffic to be forwarded.

Since the sequence of packets transmitted by a given application protocol is a pretty good indicator of the protocol itself [19], we can still use such features, albeit with some specific procedure, to identify an application protocol by observing only the tunneled SSH packets that carry it.

The procedure mentioned above includes considering the modification to the packet size due to the operation performed by SSH to tunnel traffic. In fact, the SSH encapsulation process alters the characteristics of the packet sequence generated by the original application protocol. On each SSH endpoint, a main loop collects incoming data and produces new data to forward accordingly. All TCP segments received by the same outer application during a loop are buffered into a single SSH packet that is then padded and encrypted according to the SSH protocol.

The SSH packet is sent to the lower levels of the TCP stacks and split into multiple segments if it does not suit the MSS of the SSH session. Empty TCP packets carrying acknowledgments for each of the two outer sessions do not originate corresponding data inside the inner SSH session as the three sessions are separated at the TCP level. Indeed ACK packets inside the SSH session do not add any information useful to detect the application being tunneled. For this reason we consider only TCP segments that carry data on *each* of the analyzed sessions.

The features we choose for our classification algorithms are hence the **size** and the **direction** of the encrypted packets and are gathered by observing each encrypted session at the IP-level.

<sup>1</sup>Or SSH signaling information, such as the one that transports the mandatory SSH authentication data. We will see how to deal with this in Section VI-B.

### C. From features to patterns

For each tunneling session, we extract the features from the first  $N$  non-empty packets following the SSH channel open. We pad with zeros sessions shorter than  $N$  packets. To consider the information about the direction of packets, the length of the  $i$ -th packet is transformed as follows:

$$x_i = \begin{cases} +s_i + K & \text{if } i\text{-th pkt sent by client,} \\ -s_i - K & \text{if } i\text{-th pkt sent by server.} \end{cases} \quad (4)$$

where  $s_i$  is the size of the TCP data field of the  $i$ -th packet and the constant  $K$  separates the spaces where packets traveling in the two directions lie. We take advantage of this when building the GMM and SVM models. We replace each SSH session by its equivalent pattern representation  $\mathbf{x}$  given by Equation 4; the resulting vector

$$\mathbf{x} = (x_1, x_2, \dots, x_N) \quad (5)$$

are used as input to the statistical techniques described in Section IV.

## VI. COLLECTION OF SSH-ENCRYPTED TRAFFIC: DATASET AND GROUND TRUTH

In this section we describe the procedure we used to collect the SSH-encrypted traffic at the base of our experiments, and how we derived the dataset’s “ground truth”, i.e., the application that was behind each encrypted session. Collecting traffic sessions and determining their application class is relatively easy when encryption is not used: port and payload-based analyzes can be used to assign each session to its application protocol with a relatively high precision, at least for the most common application classes for which signatures, in the form of regular expressions, have been created [20]. Unfortunately this procedure does not work when traffic is forwarded over a protocol that encrypts the tunneled sessions as is the case of SSH.

In the remaining part of this section we report the experimental set-up we used to collect real SSH-tunneled sessions along with the corresponding ground truth. The obtained data has then be used to verify the precision of the classifiers we introduced in this paper.

### A. Automatic ground truth collection: SSHgate

We collected encrypted sessions by routing the entire traffic of a given network through an SSH tunnel. To this end we developed a new application that configures SSH instances automatically and routes each new connection through a dedicated SSH process: we called it *SSHgate* [2].

The tool must run on the host designated as the SSH tunnel entry point and turns the host into a special kind of router. When a packet arrives at its network interface, it is analyzed by *SSHgate*: if the host is not the final destination of the packet and the received segment has the SYN flag turned on, the tool establishes a new forwarding SSH session with the designated SSH tunnel exit point. The SSH remote server will then forward the resulting tunneled stream to the same

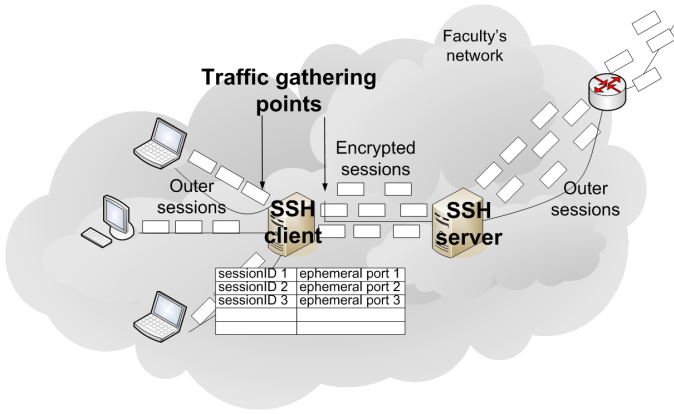


Fig. 1. Experimental set-up to gather outer and ssh-encrypted traffic.

destination address (IP, port) found on the received packet; an index for this session is stored in a session table and the packet is properly mangled to be re-injected in the network stack and tunneled inside the SSH session. Packets that do not carry SYN flag are looked up in the session table and sent to the matching session (if any), while segments coming back from each session are de-mangled and sent to the correct peer on the local network. As shown in Figure 1, for each users’ new outer session that should result in a port-forwarding request, the tool allocates a new SSH tunnel and log the address (IP, port) of the source and the destination of the original outer flow together with the ephemeral port number used by the SSH client process to connect to the SSH server.

Note that this is mimicking exactly what would happen with “regular”, non-SSHgate secure shell tunnels. In other words, the presence of SSHgate is only helping the collection of ground truth information during the experiments, but does not change the way the regular SSH implementations that we used work.

### B. Detection of channel boundaries

In order to build the traffic patterns correctly, we must filter out the SSH signaling stage that performs host and user authentication [21], [22]. As we are interested only on those packets that carry the first few encapsulated segments of the outer connection, we must collect packets only after the SSH signaling that triggers the opening of a new forwarding channel is transmitted. To this end we thoroughly analyzed the SSH protocol as implemented in the OpenSSH suite [23] and we discovered (with surprise) that the channel open message is *always* characterized by a 96 bytes long packet sent by the SSH client to the SSH server, followed by a 48 bytes long confirmation packet sent back to the SSH client. What is interesting to note is that such pair of packets appears *only* during a channel opening. Similarly, we discovered that there are only two distinctive patterns that reveal the channel close procedure. The channel is closed when both the client and the server send to the other party a channel close message and subsequently a channel eof message. All these messages are 32 bytes long. An alternative pattern is when the server sends

the channel close and the channel eof messages multiplexed into one single packet that is 64 bytes long. We used these tricks to extract from each SSH session only those packets between a channel message open and a channel message close: as already said in the previous section, we exclude empty TCP packets carrying only acknowledges.

### C. Dataset composition

Thanks to SSHgate, we easily collected the ground truth by configuring our edge router to source route a block of workstations through the SSHgate host: there was no need to configure any SSH client nor to change the routing table of any PC. The tool allowed us to even let peer-to-peer sessions pass inside the SSH tunnels as all the underneath SSH configuration is automatically done even when several sessions need to be forwarded at the same time, each one through a dedicated SSH session.

We collected all the local traffic on the SSHgate host and the encrypted traffic on the path between the SSH end-points. We exploit the information given by the tool to correlate the encrypted sessions with the outer sessions: given the ephemeral port of an SSH session we look in the log produced by SSHgate to discover the corresponding outer flow. Then, by means of payload analysis on the clear-text flow we ascertain the nature of the encrypted session.

The training data are processed as described in Sections IV-A (for GMM) and IV-B (for SVM). We built models for four protocol classes using one thousand sessions for each of them, i.e., HTTP, POP3, POP3S and EMULE: we trained each class with the same number of samples, as we do not make any assumptions on the a-priori probability of occurrence of each application in the training. We choose these protocols because they correspond to the most widespread applications in our network and are even representative of the most common traffic classes: web, P2P and mail protocols. We plan to increase the number and types of training protocols in a future work.

We also collected, in a separate timeframe, an “evaluation set” to test the accuracy of the classifiers. In addition to the protocols described above, we also captured three application protocols for which the classifiers did not receive training for: MSN over SSH, HTTPS over SSH and BitTorrent (BT) over SSH. This last set is used to verify the classifiers’ ability to recognize protocols different than those used during

over SSH	training set	evaluation set
HTTP	1000	6316
POP3	1000	447
POP3S	1000	710
EMULE	1000	1043
MSN	-	468
HTTPS	-	438
BT	-	4109

TABLE I

PROTOCOLS AND NUMBER OF ENCRYPTED SESSIONS COMPOSING THE TRAINING AND THE EVALUATION SETS.

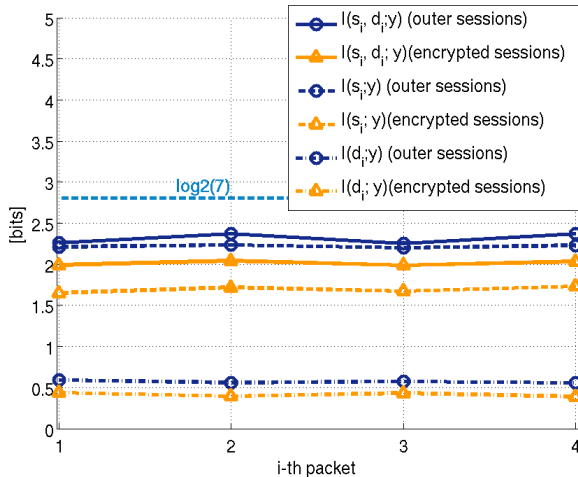


Fig. 2. Mutual information between the features extracted from the first packets of each outer and ssh-encrypted session (measured in bits).

the training phase. Overall, the evaluation set was composed of at least four hundred encrypted sessions for each of the considered protocols. We report in Table I the number of encrypted sessions composing the training and the evaluation sets.

## VII. EXPERIMENTAL RESULTS

### A. Information measures

In this Section we analyse the information carried by the statistical features extracted from each session that allows us to discriminate between sessions generated by different protocols. We measure the mutual information between the features and the protocol index  $y$ , considering the outer sessions and the same sessions encrypted.

We consider all the seven application protocols we described in Table I; therefore  $y$ , that represents the application we want to infer from the extracted features, can assume seven distinct values. We show in the Figure 2 the value  $H(y) = \log_2 7$ , that we obtain using the same number of sessions for each class. The purpose of this approach is to analyze all the different classes in a homogeneous fashion, avoiding to skew the results towards the protocols that generate the largest amount of traffic.

All the values of mutual information in the Figure are lower than  $H(y)$  representing an upper bound to the mutual information between a feature, or a vector of features, and  $y$ , that is achievable only when the protocol is perfectly predictable.

In Figure 2 we show  $I(s_i, d_i; y)$ , corresponding to the overall information we can extract from the size and the direction of the  $i$ -th non-empty packet regarding the application protocol generated it. When we consider the encrypted sessions, the  $i$ -th packet is referred to the sequence excluding the signaling SSH authentication stage. The amount of information is less in the SSH-encrypted traffic, however it is significant and it allows us to discriminate between the different protocols.

	HTTP	POP3	POP3S	EMULE	unknown
HTTP	<b>79.9</b>	–	–	0.4	19.7
POP3	0.2	<b>98.0</b>	–	–	1.8
POP3S	–	–	<b>99.2</b>	0.1	0.7
EMULE	3.3	–	0.1	<b>84.6</b>	12.0
MSN	26.3	0.4	0.4	8.1	<b>64.8</b>
HTTPS	12.6	–	14.6	0.7	<b>72.1</b>
BT	17.9	–	1.3	20.6	<b>60.2</b>

TABLE II

CLASSIFICATION RESULTS (OPTIMAL PARAMETERS) OBTAINED WITH THE GMM-BASED CLASSIFIER.

	HTTP	POP3	POP3S	EMULE	unknown
HTTP	<b>71.5</b>	–	–	0.2	28.3
POP3	–	<b>98.0</b>	–	0.9	1.1
POP3S	–	–	<b>98.9</b>	–	1.1
EMULE	0.4	–	0.1	<b>86.6</b>	12.9
MSN	15.6	–	–	16.2	<b>68.2</b>
HTTPS	0.2	–	13.2	5.5	<b>81.1</b>
BT	2.3	–	–	69.0	<b>28.7</b>

TABLE III

CLASSIFICATION RESULTS (OPTIMAL PARAMETERS) OBTAINED WITH THE SVM-BASED CLASSIFIER.

We also show the measures of the mutual information separately for the size  $s_i$  and the direction  $d_i$ . We observe that the feature  $s_i$  gives the largest contribution, its mutual information value in fact is around 2.2 bits for the outer sessions and 1.6 bits for the encrypted sessions. While  $I(d_i; y)$  is about 0.5 bits in both the traffic traces. However, the decrement of the information carried by the feature  $s_i$  is greater than  $d_i$  when we consider the SSH-encrypted sessions. The operations that SSH performs to tunnel the traffic modify the statistical characteristics of the packet sizes of a session in a more evident form than the transmission directions.

### B. Classification results

In Tables II and III we show the classification results we achieved by applying the GMM-based mechanism and the SVM-based respectively to the evaluation set. Numbers in bold in the top part of each table represent the TP rates, i.e., the portion of sessions, belonging to the training classes, that are correctly classified. The true-negative (TN) rates (the fraction of sessions correctly labeled as “unknown”) are the numbers in bold in the last column of the two tables.

In both cases, we choose to build the protocol models in a feature space of four dimensions. In other words, we consider four data packets before letting the classification algorithms make a decision. Experimental results suggest that this choice is a trade-off good enough to guarantee a high classification accuracy as well as an acceptable classification delay.

The classification results are promising: they support the idea that the IP-level information can actually be enough to break the privacy of encrypted streams, at least in terms of discovering the type of application that is behind each session. Furthermore, it proves that the behavior of a TCP session is

influenced by the application that generates the data streams, even if protected by SSH.

### C. Comparison between the GMM and the SVM approaches

1) *Precision*: The classification results in Tables II and III show that the identification of POP3 and POP3S tunneled traffic is good for both classifiers. The samples of these protocols are located in a relatively small region of the feature space (not shown here for space constraints) and therefore they can be easily identified, with TP rates above 98% on average and with a very low number of FP.

The recognition of EMULE sessions is comparable for the two classifiers, while the TP identification rate for the HTTP sessions is less accurate for the SVM classifier than the GMM-based technique. We also observe that the GMM-based classifier's ability to recognize sessions generated by protocols different from the training applications is less accurate than the SVM-based classifier in two out of three cases. The lower result is 64.8% and it is achieved in classifying MSN sessions. As for the BT traffic, the SVM classifier labels it as known traffic in around the 71.3% of cases, in contrast to the 39.8% achieved by the GMM classifier. However, the fraction of BT traffic incorrectly classified is almost always labeled as EMULE traffic by both the techniques. This outcome is encouraging and partially expected. Both the protocols are designed to accomplish the same task, i.e., enabling file sharing among peers. This leads the protocols to exhibit a similar behavior in terms of packets exchanged over the network, and both the statistical techniques reveal it.

2) *Computational complexity*: The complexity of the two classification algorithms<sup>2</sup> is similar. Given an evaluation session  $\mathbf{x}$ , the GMM-based classifier has to compute for each protocol the value that the Gaussian components take in the point where  $\mathbf{x}$  falls. We can thus compute the a posteriori probabilities of the training protocols and compare them with the threshold values, choosing the highest value over the corresponding threshold. Indeed, the complexity of the algorithm increases linearly with the number of Gaussian components and exponentially with the space dimension, i.e., the number of considered packets, because we use full covariance matrices in the GMM models.

	HTTP	POP3	POP3S	EMULE
GMM $\mathcal{N}(\cdot)$	30	39	14	6
SVM $SV_s$	323	53	28	332

TABLE IV  
NUMBER OF GAUSSIANS AND SUPPORT VECTORS.

Conversely, the complexity of the SVM-based classifier depends linearly on the number of Support Vectors of each protocol class and linearly also on the space dimension. The Gaussians centered at the Support Vectors in the SVM models have diagonal covariance matrices, with the same value  $\sigma$  for

<sup>2</sup>We refer here to the complexity of the actual classification stage, as opposed to the training phase.

all the Gaussians and for all the directions. The SVM is subject to more constraints due to the covariance matrices, therefore in building the models it needs more components in the sum of the decision functions, as we show in Table IV.

3) *Robustness*: In this paper we assume that the statistical properties of a packet located at the  $i$ -th position within a session generated by a specific protocol, characterizes the application. However, packets can be reordered, retransmitted or lost during the transmission from the source node to the classification node. Although we proved these events to be infrequent in our experiments, they do happen, and in these cases would lower the precision of the classifiers. We will study the effects of such issues on the classification results and how to face such problems in a future paper, possibly considering the sequence number field in the TCP headers as additional feature.

## VIII. CONCLUSIONS

In this paper we have analysed the SSH-encrypted traffic for discriminating sessions generated by different clear-text application protocols. We have analysed the information carried by the statistical features of the encrypted sessions, observing how much it has decreased respect to the corresponding clear-text sessions. We have also compared two supervised classification techniques for identifying the protocols that are tunneled inside SSH-encrypted traffic sessions, i.e., Support Vector Machine and Gaussian Mixture Model.

The analysis of the SSH signaling that takes place whenever an encrypted channel is opened and closed lead us to blindly discard all the packets not generated by the actual session being tunneled. Beyond the simplicity of our mechanism and the possibility to easily neutralize it, the use of deterministic patterns to characterize the boundaries of the SSH channels points out a first weakness in the OpenSSH implementation of the SSH protocol.

A further weakness is underlined by the results obtained by the application of our classifiers: we have demonstrated, under a simplifying albeit realistic hypothesis, how simple features such as the size and the direction of the packets that compose the encrypted session can be used to accurately determine the kind of application protocol tunneled inside the SSH connections. The rate of correct classification ranges between the 71.5% and the 99.2% and the rate of false-positives is very low. Both classification approaches are based on relatively simple algorithms, and should be implementable on off-the-shelf hardware.

We plan to further investigate the issues related to the session description, such as retransmitted data or packet loss, in a future work. We also plan to test the classifiers on a larger number of application protocols. Finally, we are studying how to patch OpenSSH implementations so as to make them more resilient to these types of attacks, without affecting network efficiency.

## REFERENCES

- [1] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251, IETF, Jan. 2006.

- [2] F. Gringoli, "sshgate." <http://sourceforge.net/projects/sshgate>.
- [3] C. V. Wright, F. Monrose, and G. M. Masson, "On Inferring Application Protocol Behaviors in Encrypted Network Traffic," *Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, Dec. 2006.
- [4] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Detection of Encrypted Tunnels across Network Boundaries," in *Proceedings of the 43rd IEEE International Conference on Communications (ICC 2008)*, (Beijing, China), May 2008.
- [5] L. Bernaille and R. Teixeira, "Early Recognition of Encrypted Applications," in *Proceedings of the 8th Passive and Active Measurement Conference (PAM 2007)*, (Louvain-la-neuve, Belgium), Apr. 2007.
- [6] C. Wright, F. Monrose, and G. M. Masson, "HMM profiles for network traffic classification," in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, (Washington DC, USA), October 2004.
- [7] M. Dusi, F. Gringoli, and L. Salgarelli, "A Preliminary Look at the Privacy of SSH Tunnels," in *Proceedings of the 17th IEEE International Conference on Computer Communications and Networks (ICCCN 2008)*, (St. Thomas, U.S. Virgin Islands), Aug. 2008.
- [8] T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.
- [9] A. Webb, *Statistical Pattern Recognition*. Wiley, 2nd ed., 2002. ISBN 0-470-84514-7.
- [10] A. Dempster, N. Laird, D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society*, vol. 39, no. 1, pp. 1–38, 1977.
- [11] N. Schraudolph, "Gradient-based manipulation of non-parametric entropy estimates," *IEEE Trans. on Neural Networks*, vol. 15, pp. 828–837, July 2004.
- [12] V. N. Vapnik, *Statistical learning theory. Adaptive and learning systems for signal processing, communications, and control*. New York: Wiley, 1998.
- [13] N. Williams, S. Zander, and G. Armitage, "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification," *SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 7–15, 2006.
- [14] N. Williams, S. Zander, and G. Armitage, "Evaluating Machine Learning Algorithms for Automated Network Application Identification," Tech. Rep. 060410B, Centre for Advanced Internet Architectures (CAIA), Mar. 2006.
- [15] A. Este, F. Gringoli, and L. Salgarelli, "Support Vector Machines for TCP Traffic Classification," tech. rep., Università degli Studi di Brescia, July 2008.
- [16] G. J. McLachlan and D. Peel, *Finite Mixture Models*. New York: Wiley, 2000.
- [17] C. J. V. Rijsbergen, *Information Retrieval*. Newton, MA: Butterworth-Heinemann, 1979.
- [18] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R.C. Williamson, "Estimating the Support of a High-Dimensional Distribution," *Neural Computation*, vol. 13, pp. 1443–1471, 2001.
- [19] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic Classification through Simple Statistical Fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 5–16, Jan. 2007.
- [20] "L7 Filter." <http://l7-filter.sourceforge.net>.
- [21] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol," RFC 4252, IETF, Jan. 2006.
- [22] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253, IETF, Jan. 2006.
- [23] "OpenSSH." <http://www.openssh.org>.