Using GMM and SVM-based Techniques for the Classification of SSH-Encrypted Traffic

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—When employing cryptographic tunnels such as the ones provided by Secure Shell (SSH) to protect their privacy on the Internet, users expect two forms of protection. First, they aim at preserving the privacy of their data. Second, they expect that their behavior, e.g., the type of applications they use, also remains private. In this paper we report on two statistical traffic analysis techniques that can be used to break the second type of protection when applied 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.

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.

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 III we describe how we designed the two classifiers based on GMM and SVM, while we show how to apply them to the classification of SSH tunnels in Section IV. Section V describes the dataset we used to obtain the experimental results, which are in turn presented in Section VI. Section VII addresses the comparison between the GMM and SVM statistical techniques. 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.

III. 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) [8]. 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 [9], [10], [11]. On the other hand, we preliminarily 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 encrypted traffic.

A. Gaussian Mixture Model

We express the GMM [12] parametric distribution as

\[ f(x) = \sum_{i=1}^{L} a_i N(x|\mu_i, \Sigma_i), \]

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 \{\( a_i, \mu_i, \Sigma_i \)\}\(^L\) represents all distribution parameters. Each Gaussian component \( N \) is parametrized by a mean vector \( \mu_i \) and a symmetric semi-positive definite covariance matrix \( \Sigma_i \).

Given a set of training observations \( T_S = \{x_1, \ldots, x_n\} \) belonging to the same class, the distribution parameters are computed using the Expectation Maximization (EM) algorithm [13]. This iterative method estimates the parameters of the parametric distribution by maximizing the likelihood function:

\[ l(\Phi, T_S) = \prod_{j=1}^{n} \sum_{i=1}^{L} a_i N(x_j|\theta_i). \]

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 \( x \) to one of the \( M \) available classes, we use the following threshold-based rejection schema:

\[ \arg \max_{i: f_i(x) > T_i} \{ f_i(x) \}, \quad \text{if } \exists i : f_i(x) > T_i \]

otherwise

(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 [14] metric over all the classes. Given a class, its \( F\)-measure metric is defined as:

\[ 2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall}) \]

(2)

where \( \text{precision} \) is the ratio of true-positives (TP) of the class over the sum of TP and false-positives (FP), while \( \text{recall} \) 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 [15]. Choosing a Gaussian kernel function, the SVM model for a given class is represented by the following sum:

\[ f(x) = \sum_{i=1}^{L} a_i N(x|y_{SV,i}, \sigma), \]

(3)

where \( L \) is the number of Gaussians, each one centered at a \( \text{Support Vector} y_{SV,i} \), while \( \sigma \) is the standard deviation value, that is the same for all directions and for all Gaussians of the class. The \( \text{Support Vectors} \) 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 \( \text{Support Vectors} \) 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 classifier assigns a sample \( x \) to the class whose decision function evaluated in \( x \) 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 \( f_i(x) \) are lower than the thresholds, the observation \( x \) is labeled as \textit{unknown}, i.e., it does not belong to any of the training protocols.

IV. 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 flows 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\(^1\).

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 [16], 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 \textit{size} and the \textit{direction} of the encrypted packets and are gathered by observing each encrypted session at the IP-level.

\(^1\) 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 V-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 each packet is transformed as follows:

$$s = \begin{cases} +LTCP + K & \text{if pkt sent by client}, \\ -LTCP - K & \text{if pkt sent by server}. \end{cases}$$ (4)

where $LTCP$ is the length of the TCP data field 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 given by Equation 4; the resulting vectors $x$ are used as input to the statistical techniques described in Section III.

V. 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 [17]. 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 flows 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 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 [18], [19]. 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 [20] 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 sends 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 flows pass inside the SSH tunnels as all the underneath SSH configuration is automatically done even when several flows 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 flows: 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 III-A (for GMM) and III-B (for SVM). We built models for four protocol classes using one thousand flows 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 the training phase. Overall, the evaluation set was composed of at least five hundred encrypted flows for each of the considered protocols.

VI. EXPERIMENTAL RESULTS

In Table I we show the classification results we achieved by applying the GMM-based mechanism (on the left) and the SVM-based (on the right) 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.

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.

VII. COMPARISON BETWEEN THE GMM AND THE SVM APPROACHES

A. Precision

The classification results in Table I show that the identification of POP3 and POP3S tunnelled 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 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 flows 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.

B. Computational complexity

The complexity of the two classification algorithms2 is similar. Given an evaluation session x, the GMM-based classifier has to compute for each protocol the value that the Gaussian

---

2We refer here to the complexity of the actual classification stage, as opposed to the training phase.
components take in the point where \( x \) 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.

<table>
<thead>
<tr>
<th></th>
<th>HTTP</th>
<th>POP3</th>
<th>POP3S</th>
<th>EMULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMM ( \Lambda (\cdot) )</td>
<td>30</td>
<td>39</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>SVM SV’s</td>
<td>325</td>
<td>53</td>
<td>28</td>
<td>332</td>
</tr>
</tbody>
</table>

**TABLE II**

**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 all the Gaussians and for all the directions. The SVM is subject to more constrains 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 II.

**C. 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 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**