Machine Learning Techniques for Passive Network Inventory

Jérôme François, Humberto Abdelnur, Radu State, Olivier Festor

Machine Learning Techniques for Passive Network Inventory

Jérôme François, Humberto Abdelnur, Radu State and Olivier Festor

***Abstract*—Being able to fingerprint devices and services, *i.e.*, remotely identify running code, is a powerful service for both security assessment and inventory management. This paper describes two novel fingerprinting techniques supported by isomorphic based distances which are adapted for measuring the similarity between two syntactic trees. The first method leverages the support vector machines paradigm and requires a learning stage. The second method operates in an unsupervised manner thanks to a new classification algorithm derived from the ROCK and QROCK algorithms. It provides an efficient and accurate classification. We highlight the use of such classification techniques for identifying the remote running applications. The approaches are validated through extensive experimentations on SIP (Session Initiation Protocol) for evaluating the impact of the different parameters and identifying the best configuration before applying the techniques to network traces collected by a real operator.**

***Index Terms*—fingerprinting, inventory management, syntactic tree, SVM.**

I. Introduction

ASSUMING a communication service, device fingerprinting aims to determine exactly the device version or the protocol stack implemented by a piece of equipment implementing the service. It is a challenging task which has impact on both security assessment and network management especially inventory management. Identifying the devices helps to get a detailed view of alive equipments on a network for planning future actions when needed. For example, if a new security flaw is discovered for some device types, patching them has to be fast due to zero-day attacks, but locating them is not always obvious. Besides, the fingerprinting tools can help to discover abnormal devices on a network as for instance a rogue equipment which has to be disconnected. Since device fingerprinting determines the software versions, tracking copyright infringements is another application of such techniques. Furthermore, some authentication systems check the device type like for example allowing only some specific hardphones on a VoIP (Voice over IP) network. Classical management solutions like SNMP [1] are not always applicable since some machines are often not owned by the operating company (personal or partner company devices) or their software does not provide SNMP support. Finally, fingerprinting its customers can be valuable for a company. For instance, a VoIP operator can offer additional services to its customer by sending customized advertisement based on the brand and the version of their smartphone.

Most of the current approaches for device identification are based on looking at some specific field value of the tracked protocol grammar. For instance, the SIP [2] protocol includes the device identity in the User-Agent field which can be easily omitted or modified by an attacker. Hence, new generic techniques considering the whole message are required. In this paper, every message is concerned in full and represented as a syntactic tree (section V-B) close to those which can be generated by a standard syntax analyzer. Relying on underlying differences in the content and structure of such trees, the two main contributions of this paper are :

- a new supervised syntactic fingerprinting technique which aims to precisely identify equipment (device type) (*Problem 1*),
- a novel unsupervised syntactic fingerprinting technique looking for the number of distinct device types running a given protocol and its distribution (*Problem 2*)

The second method gives general indications about the device type distribution for a given protocol and can exhibit heterogeneity or homogeneity. For instance, when a new service is deployed, proposing a support service is a real benefit for helping the users (company networks) or for doing business (operator). Hence, unsupervised fingerprinting helps to assess the complexity and the feasibility of the support service (number of distinct device versions to support). Generally, few software are supported or proposed as it is easier to provide patches and help to the users. However, many of them install other ones with additional functionalities or just because they prefer the interface for example. The number of device types used and their distribution is a good hint to evaluate the security risk because the higher the number of non supported versions, the higher becomes the risk. Moreover, these techniques are passive *i.e.*, without any interaction with the fingerprinted equipment which avoids both to be detected and an unnecessary overloading of the network and fingerprinted devices. Assuming that the majority of messages are not faked, unsupervised fingerprinting can be the foundation of the supervised system since a user can manually identify some components of the discovered clusters. Finally, the observation of the evolution of the unsupervised classification could highlight emergence or disappearing of devices.

The efficiency of these applications is directly correlated to the precision of the fingerprinting techniques. Hence, accurate classification techniques have to be employed as for example

H. Abdelnur and O. Festor are with MADYNES - INRIA Nancy-Grand Est, France, {firstname.lastname}@loria.fr

J. François, R. State are with the University of Luxembourg, {firstname.lastname}uni.lu

This work was mainly done when J. François was with MADYNES - INRIA Nancy-Grand Est, France

Manuscript received October 19, 2009, accepted July 6, 2010

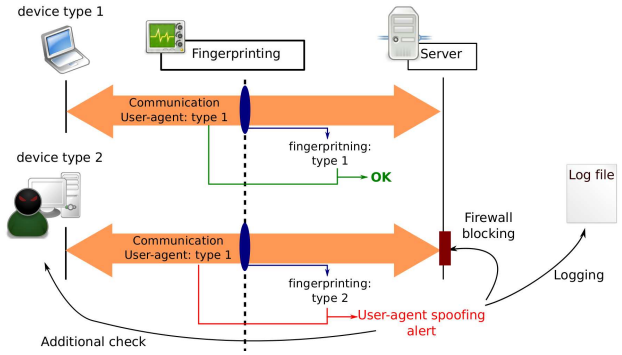


Fig. 1. Rogue device detection

machine learning methods. Our supervised fingerprinting is built on support vectors machine technique whereas the unsupervised one is based on a new adaptation of the ROCK and QROCK algorithm. Since the device representation is based on syntactic trees, the first objective is to define metrics for comparing such structures whereas classification algorithms are usually based on real values. This paper provides a study about the impact of different parameters in order to exhibit the best configurations for a real application.

The next section details some applications of device fingerprinting and the general operation of our approach. Section III formally describes the two problems. Section IV gives the details of the classification methods before introducing the device representation in section V. Section VI is dedicated to the presentation of extensive results. Related work is given in section VII before the conclusion and directions for future work.

II. Fingerprinting framework

A. Applications

This section highlights some applications that can be supported by fingerprinting techniques. Figure 1 is related to security issues since the goal is to detect rogue devices which spoof the user agent field in the messages. This field is present in many protocol messages and indicates the device type. However, it can be very easily spoofed. Since an attacker uses dedicated tools for performing an attack, the user agent should not be an usual one *i.e.* of the same type as normal users. Hence, a spoofing technique is generally employed. In figure 1, the fingerprinting tool captures the traffic between the devices and the server or between two clients as for instance in P2P networks. Considering the first communication, the fingerprinting result indicates the same type of device as the announced one. So, no counter-measure is applied. The second case shows an attacker spoofing the user agent field. By comparing this user-agent information with the automatic identification resulting from fingerprinting, the tool is able to detect the spoof and can decide to launch a counter-measure which can vary from simple logging of the event to deny access to the rogue device through interactive firewall configuration.

The fingerprinting technique has to be resistant to user agent spoofing. In this case, it can be used for various applications as for example in figure 2. In this case, network traces are frequently collected (1) for doing fingerprinting and updating a database of devices (1'). When a new patch is available for a certain device type (for example to counter a new attack or to add/correct some functionalities), the database gives recent information about the localization of devices *i.e.*, their IP addresses.

B. SIP overview

The evaluation of our fingerprinting framework is based on SIP [2] since this signaling protocol is gaining large support and the number of compliant devices is skyrocketing. SIP is a text protocol with several primitives (`INVITE`, `NOTIFY`, `REGISTER`, `ACK`, `CANCEL`...) and response codes (three digits number). The SIP specifications describe many functionalities which can be exploited by attackers as for example in [3] which highlights an authentication flaw where an attacker is able to gather the credential of a user for making phone calls. Because the attacker tool for performing the attack does not generally use the same protocol stack as the normal device, it can be detected by the fingerprinting service as explained in the previous section. This example emphasizes that a reinforced fingerprinting-based authentication SIP mechanism is possible.

C. Architecture

The architecture is depicted in figure 3. Messages are collected through SIP proxies. For each of them, the syntactic tree is constructed based on the protocol grammar. This tree represents its signature. In the case of supervised fingerprinting, two stages are needed:

- the training stage (1) : the signatures are collected and stored in a database and used for computing a classifier;
- the testing phase (2) : each new generated signature is taken as an input to the classifier to assign a specific label device type to the message.

Assuming the unsupervised case, these trees are directly grouped by computing a classifier.

Because a fingerprint of a device type is its general characterization, the proposed scheme implies the following definition : a fingerprint of a device type is the set of signatures belonging to this type in the training set. For the supervised fingerprinting, a device fingerprint is equivalent to all trees of the same device type in the training set. Then, each tree to test is a fingerprint of a device to identify. For the unsupervised technique, the fingerprint of a type is the entire corresponding cluster obtained after the classification.

Since a normal connection follows a standard sequence of messages, applying fingerprinting on the first message of each sequence should be sufficient. However, an attacker can also insert crafted packets inside a session to perform a man in the middle attack. So, our proposed architecture targets to fingerprint all messages.

III. Problem definition

Because we focus on fingerprinting, this section describes the two general problems outlined in the introduction. The

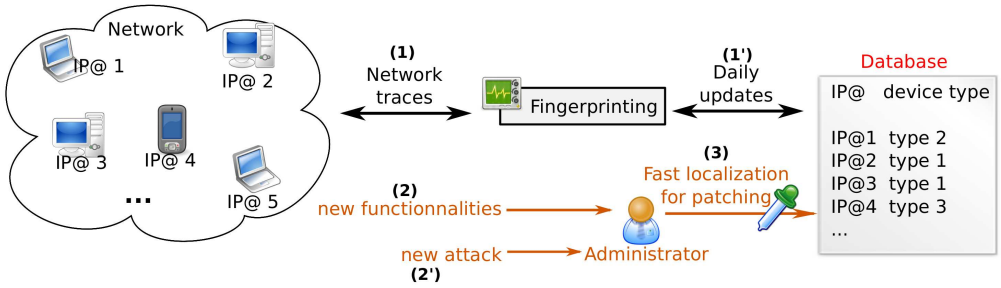


Fig. 2. Fast patching

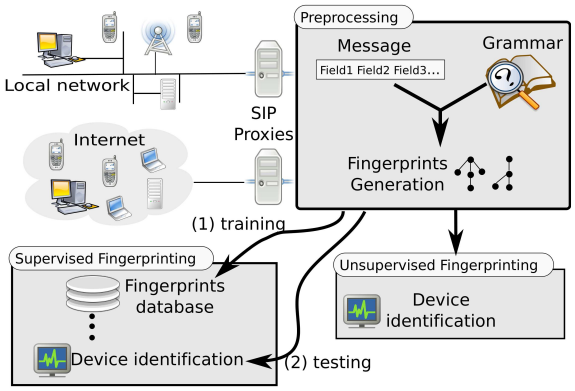


Fig. 3. Fingerprinting architecture

network is composed of several devices belonging to different types. We consider K different device types represented by the set $D = \{d_1, \ldots, d_K\}$. The traffic is collected and the messages to analyze are grouped within a testing set of N messages $T = \{t_1, \ldots, t_N\}$.

If the training stage exists (supervised fingerprinting), M messages are collected (independently from T) and labeled correctly. So, we assume the existence of the labeled training data $L \times D = \{(l_1, p_1), (l_2, p_2), \ldots, (l_M, p_M)\}$ where $p_i \in D$ and represents the label of the message l_i. Thus, each l_i is unique.

For describing the problem objectives, we suppose the existence of the function $real(t_i) : T \to D$ returning the real identifier (device type or implementation stack). Obviously, this function is not provided in reality for any message in T. So this function can be regarded as an oracle for the set T used only for describing the objectives of the following problems.

A. Problem 1: Supervised fingerprinting

The first problem aims to identify the exact types of devices which generate the messages from T. To perform this task, the system has to be trained with the training set L. So, the goal is to compute the classifier $\Omega_L : T \cup L \to D$ assigning the right device identity to each $(l_i, p_i) \in L \times D$ *i.e.*, $\Omega_L(l_i) = p_i$. The same function is then applied to each $t_i \in T$ and is expected to return $real(t_i)$, *i.e.*, being able to correctly identify the type of the device which sent the message t_i.

B. Problem 2: Unsupervised fingerprinting

In this scenario, no labeled messages are available and thus training is impossible. The messages have to be directly divided into groups by a classifier $\Psi_T : T \to \mathbb{N}$. Because no labels can be derived from a training process, the goal is to find the number of device types, *i.e.*, K, and create consistent groups containing in the optimal case only messages of a single device type. Thus, the targeted result is :

$$|\Psi[T]| = K$$

$$\forall t_i, \forall t_j,\ real(t_i) = real(t_j) \Leftrightarrow \Psi_T(t_i) = \Psi(t_j)$$

$$\forall t_i, \forall t_j,\ real(t_i) \neq real(t_j) \Leftrightarrow \Psi_T(t_i) \neq \Psi(t_j)$$

IV. Fingerprinting approaches

As previously introduced, fingerprinting is strongly correlated to classification methods which are detailed in this section. Assuming that each device to classify is represented by a syntactic tree (details in section V), the goal is to resolve the problem defined in the previous section.

A. Supervised classification, problem 1

Supervised learning techniques are potential solutions for resolving *Problem 1* since training samples are available. We chose to carry out the recent support vector machines (SVM) technique because it outperforms the classification accuracy in many domains with limited overhead [4]. SVM were already successfully used in network security monitoring and intrusion detection [5][6]. However, none of the approaches combines SVM and syntactic trees. Basically designed for two classes classification, SVM techniques were rapidly extended to multi-class problems like *Problem 1*. One-against-one classification [7] is known for providing a good accuracy with a low computational time [8].

The method strives to find a hyperplane to highly separate the data points (trees) of different classes (device types), *i.e.*, the hyperplane is as far away as possible from the data points. For the one-against-one method, an hyperplane is constructed for each pair of distinct classes as illustrated by the simple example on figure 4 where H_{i-j} is the hyperplane separating points of class i and j. Then, when the new point \$ has to be assigned to a class, its side-position from each hyperplane is computed to judge the more suitable class. Considering

the example, the following results are obtained for each hyperplane :

- H_{U-X}: \$ class is U,
- H_{O-U}: \$ class is O,
- H_{O-X}: \$ class is O.

The final decision relies on a voting mechanism where the most represented class, O, is chosen.

In most cases, the data points are not linearly separable, so they are casted in a high dimensional feature space using a mapping function φ. This function projects each initial data point into a higher dimensional space with a scalar product function defined. The main motivation is that the more dimensions are added, the more probably the data will be separable. Since the goal is to find an hyperplane, this feature space is characterized by real valued dimensions. In brief, the mapping function is formally defined as: $\varphi : T \cup L \rightarrow (R)^{dim}$ (T and L are the testing and training sets).

The exact definition of dim and the mathematical definition are dependent on the context but generally hard to determine. To counter this problem, a so called kernel trick is used. It will be detailed later.

Determining the hyperplanes is the main task. Assuming the notations introduced in previous sections, for each pair of device types $< d_l, d_p >$, the corresponding hyperplane is specified by the vector w^{lp} and a scalar b^{lp}. It has to separate and be as far away as possible from the trees belonging to d_l and d_p denoted as :

$$\begin{aligned} T_l &= \{(l_i, p_i) \in L | p_i = d_l\} \\ T_p &= \{(l_i, p_i) \in L | p_i = d_p\} \end{aligned} \tag{1}$$

Hence, the resulting problem constraints is defined as :

$$\begin{aligned} &\forall (t_i, p_i) \in \{T_l \cup T_p\} \\ &\langle \varphi(t_i) \cdot w^{lp} \rangle + b^{lp} \geq 1 - \xi_{t_i}^{lp}, \ if \ p_i = d_l \\ &\langle \varphi(t_i) \cdot w^{lp} \rangle + b^{lp} \leq -1 + \xi_{t_i}^{lp}, \ if \ p_i = d_p \end{aligned} \tag{2}$$

where ξ are slacks variables allowing some misplaced points when a total separation is not possible. For example, on figure 4, if a point O is in the surrounding of points X, it is impossible to really separate them.

The optimization problem implies that the points have to be as far away as possible from the hyperplane :

$$\min_{w^{lp}, b^{lp}, \xi_{t_i}^{lp}} \frac{1}{2} ||w^{lp}|| + C \sum_{(t_i, p_i) \in \{T_l \cup T_p\}} \xi_{t_i}^{lp} \tag{3}$$

where C is a constant representing the trade-off between the minimum classification errors and maximal margins.

The function φ is essential but defining it is often hard. That is why the kernel trick is usually exploited by defining a kernel function $K(x, y)$ [9] directly computable between two trees and also equals to $\langle \varphi(x_i).\varphi_(x_j) \rangle$. Then, the optimization problem is turned into its dual by introducing the Lagrangian for computing the vector w^{lp} :

$$max \sum_{(t_i, p_i) \in \{T_l \cup T_p\}} \alpha_{t_i}^{lp} - \frac{1}{2} \sum_{\substack{(t_i, p_i) \in \{T_l \cup T_p\} \\ (t_j, p_j) \in \{T_l \cup T_p\}}} \alpha_{t_i}^{lp} \alpha_{t_j}^{lp} \rho_{t_i}^{lp} \rho_{t_j}^{lp} K(t_i, t_j) \tag{4}$$

subject to:

$$\begin{aligned} \sum_{(t_i, p_i) \in \{T_l \cup T_p\}} \alpha_{t_i}^{lp} \rho_{t_i}^{lp} &= 0 \\ 0 \leq \alpha_{t_i}^{lp} \leq C, \ (t_i, p_i) &\in \{T_l \cup T_p\} \end{aligned} \tag{5}$$

with :

$$\rho_{t_i}^{lp} = 1 \ if \ (t_i, p_i) \in T_L, -1 \ if \ (t_i, p_i) \in T_P \tag{6}$$

The scalar b^{lp} is calculated thanks to the support vector trees (SV^{lp}) which corresponds to the points on the border of each group mathematically defined as the trees t_{sv} such that $\alpha_{t_sv}^{lp} \neq 0$:

$$b^{lp} = \frac{1}{|SV^{lp}|} \sum_{t_i \in SV^{lp}} (\rho_{t_i}^{lp} - \sum_{(t_j, p_j) \in \{T_l \cup T_p\}} \alpha_{t_i}^{lp} \rho_{t_j}^{lp} K(t_j, t_i)) \tag{7}$$

Once the training phase has solved this optimization problem, each tested tree $l_m \in T$ is given as the input of the decision function :

$$f_{lp}(l_m) = \sum_{t_i \in \{T_l \cup T_p\}} \alpha_{t_i}^{lp} \rho_{t_i} K(t_i, l_m) + b^{lp} \tag{8}$$

Then, the sign of the result indicates if l_m belongs to d_p or d_l. Since, only one hyperplane is defined for each pair of class, these functions are symmetrical $f_{lp} = -f_{lp}$. So only $\frac{K(K-1)}{2}$ hyperplanes and functions have to be found out.

Finally, the classifier Ω_L formally defined in Problem 1 represents the voting scheme. Each decision function chooses a class among two and the classifier Ω_L returns the most chosen class. Assuming the distribution $V(t_x = k) = \frac{|\{f_{dk}, f_{dk}(t_x) > 0\}|}{|\{f_{lk}\}|}$, the identification result of the tree t_x corresponds to $\Omega_L(t_x) \arg\max_k V(t_x = k)$.

B. Unsupervised classification, problem 2

1) ROCK and QROCK: Whereas our device representation is based on syntactic trees which can be viewed as categorical data, most well known techniques such as K-means, K-medoids or density based algorithms are suited for numerical values [10]. Therefore, new kind of unsupervised approaches dedicated to categorical data can be found in the literature as for instance the ROCK algorithm [11]. This algorithm is based on a graph representation where two nodes are linked if they share at least one common neighbor. Two points are neighbors if their inter-distance is less than a threshold τ (the distance between two trees is defined in section V). It is an agglomerative clustering technique and so each unique point is a cluster at the beginning. Then, the clusters are grouped together based on a score measure which measures the linkage degree (the number of shared neighbors) comparing with the estimation of the maximal number of possible shared neighbors.

Figure 5 highlights the results of main types of clustering. Figure 5 points out the bad accuracy of medoid clustering methods which group points around another one. The main disadvantage is that these techniques assume similar points

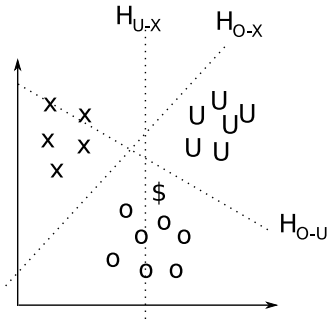


Fig. 4. SVM one-to-one classification, $ is a new point to assign

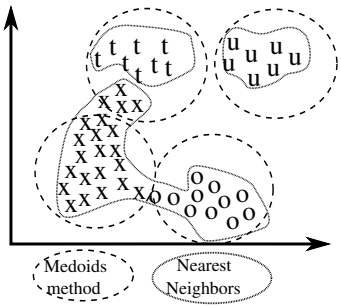


Fig. 5. Common classification problems

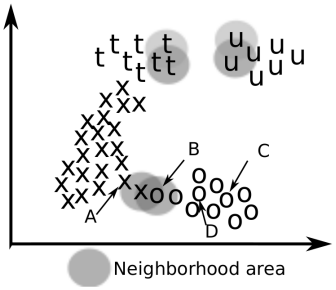


Fig. 6. Rock classification

distributed within a common shape (spherical most of the time) close to a medoid. Other well-known techniques consider each point individually. For example, the nearest neighbors technique results is plotted in figure 5: the clusters of the pair of closest points are merged until the corresponding minimal distance is higher than a threshold. The main advantage is the discovery of irregular shapes of clusters. For example, in figure 5, the distinct shapes of clusters "t", "u" and "x" are easily distinguished because their closest nodes are well separated. However for "x" and "o", the boundary points are very close and a classic approach merges them. The ROCK algorithm looks for points sharing common neighbors which is not the case for these points as shown on figure 6. However in this case, the points A and B should be linked because they share a common neighbor. That is why a score measure is introduced to join two clusters with the maximum number of neighbors. Here, the algorithm prefers to join C and D rather than A and B. Hence, the ROCK algorithm is capable to discover the right clusters. Other such methods exist like CURE for example where each cluster is fixed by a limited number of points, so it is a tradeoff between one center and all points. Density based clustering techniques such as DBScan are close to ROCK which is well suited for categorical data like trees. We refer to [10] for a good overview of these algorithms and their use cases.

However, ROCK is heavy computational and a derived version, QROCK, was proposed in [12]. The authors of QROCK observed that in many cases, the computed clusters are equivalent to compute the connected components of the created graph. Thus, QROCK considers that each connected component is a single cluster. Hence, the algorithm becomes very simple and is executed very fast. The main disadvantage is that the points A and B in figure 6 will be joined due to their unique common neighbor. In fact, QROCK does not take in account the neighborhood density measured by the score measure.

2) Compromise: The limitations of ROCK and QROCK logically imply to choose a fair trade-off between them with following ambitions :

- keep the advantage of the neighborhood density (ROCK),
- avoid too much computational metric (QROCK).

The first idea is to choose a simple score measure. The most simple should be to sum all links between each pair of clusters but the authors of ROCK advice against it. In fact, it often entails the creation of a single or few big clusters because the bigger a cluster is, the more neighbors it has. We present a new simple metric for evaluating the score measure between two clusters: the maximal number of shared neighbors between any pair of two nodes from each cluster. Assuming, two clusters C_i and C_j, the score measure between the clusters is:

$$good(C_i, C_j) = max_{p_t \in C_i, p_l \in C_j} \#neighbors(p_t, p_l) \quad (9)$$

where $\#neighbors(p_t, p_l)$ returns the total number of shared neighbors between p_t and p_l. This metric is very simple to compute because the distance between two points does not vary whereas the original goodness of ROCK is updated during the clusters merging since the metric is based on all shared neighbors between all points of two clusters. Moreover, estimating the total number of possible neighbors for normalizing this value against the size of the cluster is unnecessary with the new metric.

The clusters are merged until this new score reaches a threshold γ. Thus, the clustering has to join two points p_1, p_2 for which $good(p_1, p_2) > \gamma$.

Theorem 1: The results of the ROCK algorithm based on the score measure $good$ defined in equation (9) is independent from the order of merging points.

The proof is direct as the definition of $good$ is only dependent on the points themselves and not on the clusters, *i.e.,* other points. This theorem is very important as there is no need to order points following the decreasing value of the goodness measure like in ROCK. Thus, the overall complexity is significantly reduced. Besides, it corresponds to the QROCK algorithm with one additional constraint. In fact, the graph links are weighted by the number of shared neighbors and the objective is to determine the connected components of vertices with weighted links equal to at least γ to keep the neighborhood density as a valuable information. Hence, the algorithm design is straightforward and split into two main functions :

- the graph construction based on the neighborhood computation;

- the computation of connected components.

The first step is executed by algorithm 1 where L_{ij} (the adjacency matrix) is the number of shared neighbors between i and j. In fact this algorithm iterates over all pairs of points (trees in our case). When two of them are neighbors, the algorithm considers one as the shared neighbors and looks for its other neighbors to update the weighted adjacency matrix (loop of the line 4).

Algorithm 1 Link initialization

$T = \{t_1, \ldots, t_N\}$ a set of trees
D_{ij} is the distance between the tree t_i and t_j
τ the maximal distance between two neighbor trees
L_{ij} the number of neighbors between the tree t_i and t_j initialized to 0

```
1: for i ← 1 to N do
2:   for j ← 1 to N do
3:     if D_ij < τ then
4:       for k ← 1 to N do
5:         if D_ik < τ then
6:           L_kj = L_kj + 1
7:         end if
8:       end for
9:     end if
10:  end for
11: end for
```

Then, algorithm 2 computes the connected components having links with at least a weight of γ neighbors. At the beginning, each tree is associated to a label set to FALSE indicating that the tree is not in a cluster yet. The algorithm iterates over all trees searching non visited ones and creates a new cluster. Then, the recursive clustering function is applied on the trees that share the minimum number of neighbors with the initial tree in order to add them and so on.

Assuming the additional function $ind(t)$ returning the index of the cluster containing the tree t, the classifier of the problem 2 is defined by $\Psi_T(t) = ind(t)$.

3) Appearing or disappearing devices: A special kind of application of our approach is the detection of abnormal equipments as for example rogue equipment. However, detecting that main devices of a certain type disappear is also important since it can be caused by a dedicated attack against this type. Assuming an initial set of messages which can be viewed as a kind of "training set" without labels and containing all possible types, the clusters obtained are denoted by the function Ψ_0 with Ψ_t representing the clusters composition at time t. Theorem 1 indicates also that the assignment of messages is independent from all others. Hence, Ψ_t is constructed from Ψ_0 by adding syntactic tree of messages arriving between time zero and t. Concerning a long period of time, a type disappearance is detected if no new messages are assigned to a cluster and the identification of rogue equipment is pointed out by the creation of new clusters. This task can be done with any classification algorithm but the proposed algorithm is able to achieve it incrementally without reconstructing all the clusters at each time t.

Algorithm 2 clustering

$T = \{t_1, \ldots, t_N\}$ a set of tree
L_{ij} the number of neighbors between the tree t_i and t_j
$init(t)$ creates a cluster with only the tree t (assign an index to the cluster)
$c.add(t)$ add the tree t to cluster c
$Label_i$ indicates if t_i is already assigned to a cluster and is initialized to 0

```
1: for i ← 1 to N do
2:   if not Label_i then
3:     c = init(t_i)
4:     Label_i = TRUE
5:     for j ← 1 to N do
6:       if i ≠ j and L_ij > γ and Label_j = FALSE then
7:         clustering(j,c)
8:       end if
9:     end for
10:  end if
11: end for

12: clustering(k,cluster):
13: Label_k = TRUE
14: c.add(T_k)
15: for j ← 1 to N do
16:   if k ≠ j and L_kj > γ and Label_j = FALSE then
17:     clustering(j,c)
18:   end if
19: end for
```

V. Attributed trees

Problems of section III are resolved thanks to methods of section IV. This section details the syntactic representation of devices and how to compute the similarity or distance between two devices in order to apply these methods.

A. Distances

Our techniques use the metrics defined in [13]. This section gives an overview of the theory. An attributed graph is defined by the tuple $G = (V, E, \alpha)$ where V are the different nodes, E the different edges and α is a function such that $\alpha(s)$ gives some characteristics about the node s. A tree is a special graph without cycle. Two trees T_1 and T_2 are considered isomorphic if there exists a bijection ϕ mapping every node in T_1 to every node in T_2 while keeping the same structure (the nodes are connected in the same way). In fact, the nodes are generally labeled or defined by some characteristics but a node can be mapped to a node with a different label. The trees have a subtree isomorphism ϕ if there exists two subtrees T'_1 and T'_2 which are isomorphic. Their similarity is measured as :

$$W(\phi, T'_1, T'_2) = \sum_{u \in T'_1} \sigma(u, \phi(u))$$

where σ is the comparison function between the characteristics (α function) of two nodes defined by the user. Furthermore, the goal is to determine $W(\phi_{12})$ which is the maximum similarity

```
Message =  Request  SP *Header SP 0*1Body
Request = Invite / Notify / Cancel
Invite = "INVITE"
Cancel = "CANCEL"
Notify = "NOTIFY"
Header = Accept / Date / Call-id / User-Agent
Body = *Alpha
Alpha = %x41-5A / %x61-7A       ; A-Z / a-z
HCOLON =  *SP ":" *SP
SP =  %x20                      ; space
Accept = "Accept" HCOLON *Alpha "."
Date = "Date" HCOLON *Alpha "."
Call-Id = "Call-Id" HCOLON *Alpha "."
User-Agent = "user-Agent" HCOLON *Alpha "."
```

Fig. 7. Grammar

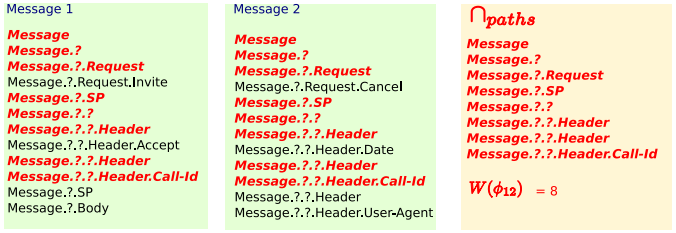


Fig. 8. Intersection of ancestor paths

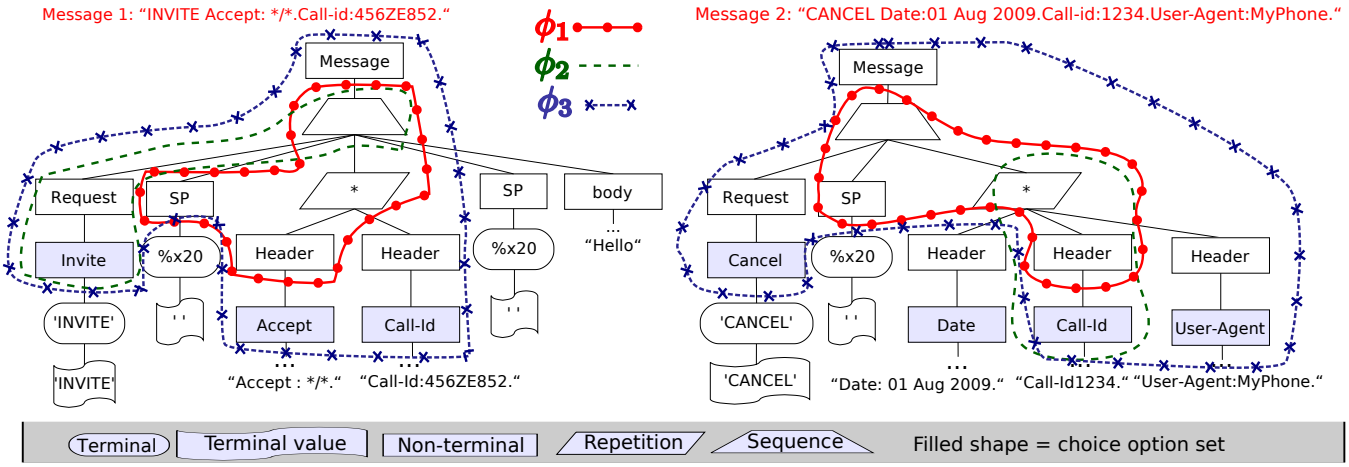


Fig. 9. Syntactic trees of 2 messages

among all pairs of isomorphic trees of T_1 and T_2:

$$W(\phi_{12}) = \max_{(\phi,T_1',T_2')\in R} W(\phi, T_1', T_2')$$

where R is the set of all possible isomorphisms and the corresponding subtrees.

Although classical techniques compute the similarity between two trees by counting the number of transformations (delete, add or substitute) required to transform the trees into isomorphic ones, the authors of [13] emphasize that resolving this problem is NP-complete unless adding some specific constraint (nodes ordering for instance) to get a polynomial-time complexity. Hence, they propose to define four novel distance metrics (two normalized and two non-normalized) between trees leading also to a polynomial complexity. The normalized metric are advantageous because they can be interpreted in an equivalent way even if the context is not the same (different devices, other protocol) whereas the non-normalized metric will exhibit great different values. However, it is impossible to know a priori which kind of distance is better in terms of accuracy. Thus, one distance metric of each type is kept for supervised and unsupervised fingerprinting. After preliminary evaluations, three of them appear to be suitable:

$$dist_1(T_1,T_2) = |T_1| + |T_2| - 2W(\phi_{12}) \ (non-normalized) \quad (10)$$

$$dist_2(T_1,T_2) = 1 - \frac{W(\phi_{12})}{max(|T_1|,|T_2|)} \ (normalized) \quad (11)$$

$$dist_3(T_1,T_2) = 1 - \frac{W(\phi_{12})}{|T_1| + |T_2| - W(\phi_{12})} \ (normalized) \quad (12)$$

where $|T|$ is the number of nodes of the tree T.

B. Syntactic trees

A syntactic tree is an attributed tree built from a message sent by a device and the Augmented Backus-Naur Form (ABNF) [14] protocol grammar. The figure 7 shows a partial grammar of a simple protocol (far from SIP). The non-terminal elements are those which can be derived into other ones (Message, Request) contrary to terminals representing a fixed sequence of or a range of possible characters (terminal values are real values in the message). The elements prefixed by "*" are repeated whereas those separated by "/" are alternatives. Otherwise, the different elements form a sequence.

Thus, each message is mapped to a syntactic tree like in figure 9. A node is created from each terminal value and linked to a parent node representing the sequence, the repetition or the non-terminal from which it is derived. Figure 9 shows two partial syntactic trees.

The syntactic trees are rooted. Thus, two trees are isomorphic if the relationship between parent and child nodes is also kept. Furthermore, terminal values are not taken into account because the containing information is highly dependent of a specific session (call-id, date...).

Some potentially large structure can be derived for many grammar rules as for example the construction of a character sequence built by the expression `*Alpha`. Thus, two subtrees with different `Request` or `Header` branches can contain

such a structure whereas their meaning is probably different. Hence, these relatively large structures could bias the similarity measure. The solution is to consider the path of a node to evaluate their similarity. The path is all the nodes between the root node and the considered node. Therefore, the characteristics of a node n defined by $\alpha(m)$ is the tuple $<name_m, path_m>$ with $path_m$ the path. The name of a node is its non-terminal name or "?" otherwise (sequence or repetition). We propose a binary similarity (σ) between two nodes imposing that two similar nodes have to share similar ancestor nodes, *i.e.*, the same path, and the same name. Assuming that par_n returns the parent node of n and r the common root of the trees, the similarity between two nodes u and v can be totally defined as:

$$\sigma(u,v) = \begin{cases} 1 & if\ u = r \wedge v = r \\ 1 & if\ name_u = name_v \wedge \sigma(par_u, par_v) = 1 \\ 0 & else \end{cases} \quad (13)$$

If messages can be derived from different first rules, adding a generic root node r is feasible but leading to a similarity equal to zero. Three subtree isomorphisms are represented in figure 9. The subtrees associated to the first ones contain exactly the same nodes and so $W_{\Phi_1} = 4$. Because sequence and repetition are equivalent in the ancestors path (question mark), the second isomorphism generates two trees sharing one similar node. However, $W_{\phi_2} = 0$ due to different ancestors path. Finally, $W(\phi_3) = 8$ because the subtrees are the same except for two nodes (`Accept` and `user-Agent`). The `Call-Id` match because there is no order on the nodes.

Though, the first isomorphism is clearly suboptimal as the subtrees are not rooted on the global root node while the pair of nodes share the same ancestors. Hence, finding the isomorphism candidates has to consider the paths of all nodes of a tree as illustrated in figure 8. The creation of the lists containing these paths can be done easily during the creation of the trees. The optimal isomorphic subtree is built from all shared paths by the messages. Thus, the subtrees are the intersection $\cap_{paths}$ of similar paths calculated by the algorithm 3 whose design is straightforward. Indeed, one iteration loops over all paths of the first tree t_1 and looks for the same path in the second one t_2. Line 10 is extremely important for avoiding to take in account the same path twice. For instance, inverting the messages in figure 8 implies three paths `Message.?.?.header` without this line. Since this algorithm iterates over each path of t_2 for each path of t_1, the complexity is in $O(t_1 t_2)$

Because all paths are rooted on the same node, the prefix of each path (all nodes except the last one) always equals another one. Hence, the similarity is exactly the number of elements in the intersection[1] : $|\cap_{paths}|$. Thus, the similarity between the example messages is eight.

Since the fingerprinting is based on structural differences between syntactic trees, it is suited for protocols whose specification allows some freedom about the message construction (optional fields, order of fields...). Since the user-agent field

[1]This is not a mathematical intersection since a path can be represented several times

Algorithm 3 similar_paths(t_1,t_2)

$res = [\,]$ is the intersection of shared paths initialized to an empty list
$paths(t)$ return the paths list of the tree t
$l.add(e)$ adds e to the list l
$l.remove(e)$ remove e from the list l
$len(l)$ is the length of the list l

```
1:  c1 = paths(t1)
2:  c2 = paths(t2)
3:  for c ∈ c1 do
4:      i ← 1
5:      bool ← TRUE
6:      while bool ∧ i < len(c2) do
7:          if c = c2[i] then
8:              bool = FALSE
9:              res = res.add(c)
10:             c2.remove(c)
11:         end if
12:         i ← i + 1
13:     end while
14: end for
```

or the flat content (characters) of the messages can be easily altered, the syntactic structure included cannot. Considering the toy example of figure 9, the fingerprinting techniques are usually based on some fields values as for example [15] which analyzes the bad randomness of the call-id field of certain devices. Therefore, an attacker can easily modify this field by replacing the value directly in the message in order to counter the fingerprinting technique. However, the syntactic tree structure is not altered and so the syntactic fingerprinting techniques are still able to identify the attacker device.

C. Adapation for fingerprinting

The distance adaptation for SVM based fingerprinting (problem 1) is compulsory because the kernel function is a similarity measure. For the normalized metric, the definition is straightforward as the similitude is equivalent to one minus the distance. For the non-normalized metric, we derived a kernel with a form close to the Gaussian one. Hence, two kernels functions are defined:

$$K_1(T_1,T_2) = e^{-0.01 dist_1(T_1,T_2)} \quad K_2(T_1,T_2) = 1 - dist_3(T_1,T_2)$$

For the unsupervised fingerprinting, the two metrics chosen for testing this new algorithm are $dist_1$ and $dist_2$. The latter one is directly applied but we do a simple transformation on $dist_1$ for having a normalized value between 0 and 1:

$$dist'_1 = 1 - e^{-0.01 dist_1}$$

The coefficient of 0.01 for $dist'_1$ and K_1 was tuned thanks to our first experiments. This manual normalization helps to interpret identically this metric and so to not reconfigure our fingerprinting algorithms each time. However it introduces this new parameter which can limit the usage of the non-normalized metric.

The next section about experimentations is based on these adaptations of distance metrics but they will be mentioned as their original notation ($dist_1$, $dist_2$ and $dist_3$) for improving the readability and the comparison between results.

VI. Experimentations

A. General description

The following experiments aims to evaluate our fingerprinting scheme. Hence, the types assigned automatically to the devices will be checked in order to compute several metrics for evaluating the performance (the next section describes the different metrics used). The first part of the experiments is based on a small dataset in order to test our system with many distinct configurations. Thus, the impact of the different parameters can be studied in order to determine the best ones for performing the fingerprinting with a real operator in a second step including only section VI-F.

This includes two kinds of parameters:

- those which can be changed easily when applied to a real dataset: algorithm parameters or distance used;
- those which are constrained by the real environment. In fact, this includes only one parameter, the training percentage, which reflects the amount of available data. Hence the goal of the corresponding experiments is not to find out the best value to set for this parameter but to indicate the minimum training percentage which is needed for guarantying a minimal quality results.

In brief, the experiments of the first part aim to determine which distance is the best suited for fingerprinting, what is the impact of the training percentage, what is the impact of ROCK/QROCK adapted algorithms on the performance in order to determine the best configurations. Since the unsupervised fingerprinting does not provide convincing results, we propose two improvements.

We also provide a short experiment to show the benefits of the syntactic structure for improving the classification.

B. Metrics

Standard metrics for classification assessment presented in [16] are adapted to our terminology introduced in section III (N is the total number of messages which are classified and K is the number of distinct device types). Assuming x_d, the number of trees corresponding to a particular device type $d \in D$, y_d the number of trees classified as d, $z_{d_2 d_1}$ the number of trees of types d_2 which were classified as d_1, the sensitivity evaluates the number of trees of a given type d which were assigned to the right cluster:

$$sens(d) = z_{dd}/x_d \tag{14}$$

The specificity of a device type d measures the proportion of trees of this type in the corresponding cluster:

$$spec(d) = z_{dd}/y_d \tag{15}$$

The overall metric name accuracy is the proportion of trees which are assigned to the correct type:

$$acc = \sum_{d \in D} z_{dd}/N \tag{16}$$

Device Name	#mesg	height			#nodes		
		Max	Min	Avg	Max	Min	Avg
Asterisk_v1.4.21	1081	28	23	25	2517	883	1284
Cisco-7940_v8.9	168	25	23	24	2784	812	1352
Thomson2030_v1.59	164	28	23	24	2576	793	1391
Twinkle_v1.1	195	25	23	23	2457	805	1299
Linksys_v5.1.8	195	28	23	25	2783	852	1248
SJPhone_v1.65	288	30	23	24	2330	951	1133

TABLE I
Testbed dataset – Tree statistics

Furthermore, the average sensitivity and specificity value is easier understandable than multiple values:

$$\begin{aligned} avg\_sens &= \sum_{d_i \in D} sens(d_i)/K \\ avg\_spec &= \sum_{d_i \in D} spec(d_i)/K \end{aligned} \tag{17}$$

Assuming the following distributions $\mathbf{X} = x_i/N$, $\mathbf{Y} = y_i/N$, $\mathbf{Z} = z_{ij}/N$, the mutual information coefficient (IC) is an entropy based measure defined as :

$$IC = \frac{H(\mathbf{X}) + H(\mathbf{Y}) - H(\mathbf{Z})}{H(\mathbf{X})} \tag{18}$$

where H is the entropy function. This ratio varies between 0 and 1 (perfect classification) and is a good complementary metric from the overall accuracy because it indicates if the accuracy value is not only due to one or few over-represented classes. For example, assigning all messages to one class can allow to reach 80% of accuracy if 80% of data points are of the same type. However, this case implies $IC = 0$. Hence, this coefficient reflects the sensitivity and the specificity and is even more severe than them.

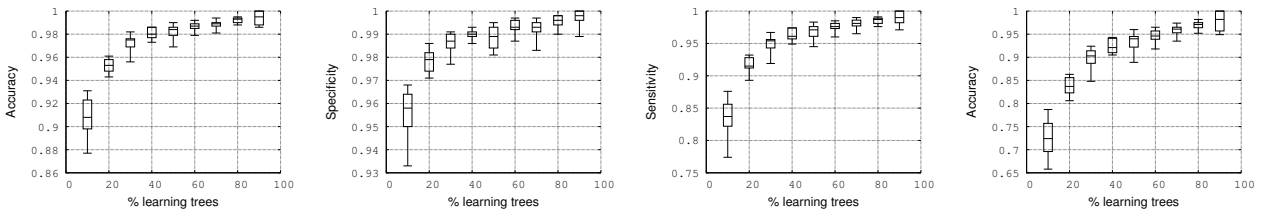
Although the supervised classification creates one labeled cluster per device type which contains testing trees, the unsupervised classification can create an arbitrary number of clusters. Even if labeling unsupervised cluster is not done in reality, the classification assessment process begins by labeling each cluster with the most represented device version in the cluster. Then, only the largest cluster of each type is kept and the rest of the trees are assigned to an artificial garbage cluster. However, evaluation of the mutual information coefficient with a garbage cluster is meaningless. So, the F-score is another overall possible metric from the information retrieval domain [17]:

$$F - score = \frac{2 \times avg\_sens \times avg\_spec}{avg\_spec + avg\_sens} \tag{19}$$

Like the mutual information coefficient, F-score is a combined measure from sensitivity and specificity but does not reflect the messages distribution. However, if all messages are affected to few classes, this score will be very low too.

C. Dataset

The main dataset having characteristics summarized in table I was generated from our testbed with 6 device types (softphones, hardphones and proxy) with a total number of 2091 messages. The syntactic trees are very big, their heights are close to 30 and the minimal number of nodes in a tree is



(a) Accuracy (b) Specificity (c) Sensitivity (d) Mutual information coefficient

Fig. 10. Supervised fingerprinting, distance $dist_1$

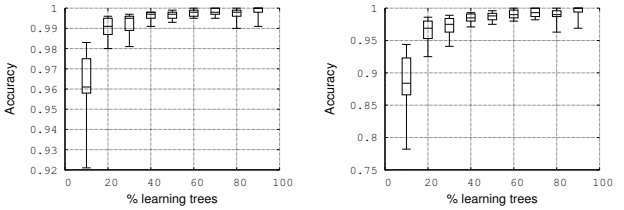
more than 800. This is due to a huge specification as the ABNF grammar [14] contains around 500 rules. Therefore providing a real tree in the paper is impossible. The following message is a real SIP message generated on our testbed and emphasizes the huge size difference with a toy example such as messages from figure 9:

```
INVITE sip:941@192.168.0.5 SIP/2.0
Via: SIP/2.0/UDP 192.168.0.2:5060;branch=z9hG4bK71a6058b
From: "Cisco 7940" <sip:7940@192.168.0.5>;tag=001ae2bc8b7c
000313f0813f-0635efb4
To: <sip:941@192.168.0.5>
Call-ID: 001ae2bc-8b7c0003-7f5491ce-2990dcf3@192.168.0.2
Max-Forwards: 70
CSeq: 101 INVITE
User-Agent: Cisco-CP7940G/8.0
Contact: <sip:7940@192.168.0.2:5060;transport=udp>
Expires: 180
Accept: application/sdp
Allow: ACK,BYE,CANCEL,INVITE,NOTIFY,OPTIONS,REFER,
REGISTER,UPDATE
Remote-Party-ID: "Cisco 7940" <sip:7940@192.168.0.5>;
party=calling;id-type=subscriber;privacy=off;screen=yes
Supported: replaces,join,norefersub
Content-Length: 273
Content-Type: application/sdp
Content-Disposition: session;handling=optional

v=0
o=Cisco-SIPUA 3326 0 IN IP4 192.168.0.2
s=SIP Call
t=0 0
m=audio 21382 RTP/AVP 0 8 18 101
c=IN IP4 192.168.0.2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
```

D. Supervised classification

1) Training percentage: The first experiment evaluates the efficiency of our supervising method in parallel with the proportion of extracted trees for the training process (training percentage). In fact, the messages are randomly selected and each experiment is run ten times to improve the fairness of our results. Considering distance $dist_1$, the figure 10 plots the accuracy metrics using a quartile representation. The extrema values are plotted and the box delimits 50% the observations with the median value as the horizontal bar. The rest of the observations are outside the box (25% below, 25% above). The overall accuracy shown in figure 10(a) highlights that our approach is very effective because with only 10% of training, the accuracy is concentrated around 90%. Obviously, increasing the number of training sample messages improves the accuracy and with 20% the limited range of quartiles



(a) Accuracy (b) Mutual information coefficient

Fig. 11. Supervised fingerprinting, distance $dist_3$

indicates stable results. So, expecting an accuracy of 95% is really viable. The average sensitivity illustrated in 10(b) means that the good accuracy is not only due to some well classified classes since this value reaches 90% with a training percentage of 20% or more.

The specificity is high also meaning that the misclassified messages are scattered among the different device types. The figure 10(d) summarizes the previous observations by amplifying the lowest specificity and sensitivity value. Hence, this figure confirms that the information coefficient is clearly more severe than other metrics. That is why the next experiments do not detail sensitivity and specificity values.

2) Distances: The distance $dist_3$ is normalized contrary to $dist_1$. Such a distance is generally easier to use since many techniques like SVM can be directly applied without considerable tunable transformation. The efficiency of the identification is clearly improved and outperforms the previous ones. With only 20% of messages used for the training, the accuracy is close to 99% in most cases. The specificity and sensitivity are also very high and illustrated here through the mutual information coefficient in figure 11(b). In fact, obtaining similar results with the previous distance needs to raise the training percentage. For instance, a percentage of 80% with $dist_1$ is equivalent to 20% with $dist_2$.

To summarize, SVM-based supervised fingerprinting is very effective with the normalized distance $dist_3$.

Figure 12 is the intensity coded distance matrix for $dist_3$. Devices are grouped by types represented by black lines on the axis (they are in the same order than in table I). Distance between messages x and y is represented by the intensity of the color of the cell (x, y). The darker the cell is, the lower the distance is. Hence, $dist_3$ is clearly able to detect group of messages especially for the first one which is well distinguished. Other black block structures are highlighted but messages from the same device type are clearly not always in such blocks. Even $dist_3$ is able to construct group of messages,

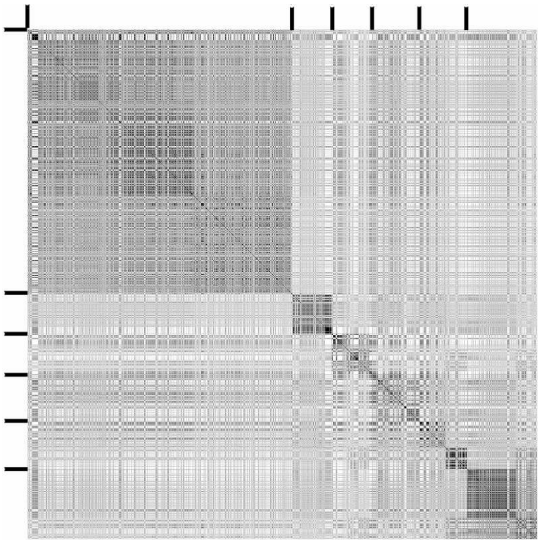


Fig. 12. Intensity coded distance matrix for $dist_3$

applying an efficient classification technique is also necessary for taking benefit from that as for example SVM.

3) Advantage of the syntactic structure: This short experiment highlights the benefit from the syntax structure compared with the flat content. A common metric for measuring the similarity between two messages is the Levenshtein distance [18]. It is also known as the edit distance because it computes the number of needed transformations (insertion, deletion or substitution) for transforming the first text message into the second one. Figure 13 reports the same results as figure 11(a) and shows also the accuracy of the application of the Levenshtein distance. Using the syntactic representation of messages, the accuracy improvement increases between 30 and 50% which clearly emphasizes the relevance of the syntactic tree representation.

E. Unsupervised classification

Applying unsupervised classification on individual messages was not successful (about 60%). This is mainly due to notable differences between messages targeting different goals (with different types) even emitted from the same kind of device. Hence a message for making a phone call or registering are highly dissimilar. With supervised classification, the training process capturing different kinds of messages for the same device type counters this problem.

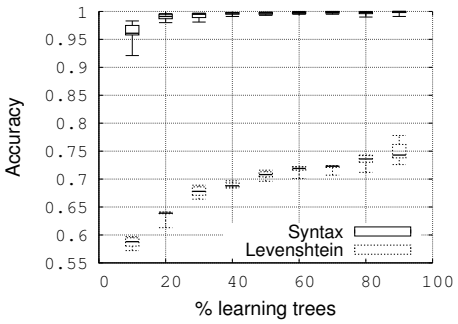


Fig. 13. Syntactic tree advantage, accuracy with distance $dist_3$

Thus, we propose two methods for improving the classification accuracy :

- identifying the kind of a device based on only one type of messages (*e.g.* INVITE),
- creating small clusters of messages sharing the same source IP address and source port within a small interval of time ρ (few seconds). This assumption is realistic as these characteristics will not change frequently for a piece of equipment.

1) Grouping messages: In the first experiment, ρ is set to 5 seconds and the goal is to determine what are the best parameters of the new version of the ROCK algorithm :

- τ : the maximal distance between two neighbors,
- γ the minimum number of shared neighbors between to messages for merging them.

With $\rho = 5$, an average of 2.8 messages are grouped in the same cluster beforehand. Except in four cases highlighted by boxed values in table II, all the different kinds of devices are discovered. The shading key helps to rapidly discard bad configurations like the pale column ($\tau = 0.01, 0.15, 0.2$) highlighting the great impact of τ. Thus, the accuracy is not a monotonic function of τ. In the same way, it is not a monotonic function of γ and 87% of messages are correctly classified by using a neighbored distance of 0.1 and a minimal of ten shared neighbors for grouping two messages. Moreover, it is ten points better than the best result of the first row which is equivalent to the QROCK algorithm (one shared neighbor only). The corresponding high value of the F-score in table III indicates that this result is not only due to few device types rightly identified but also to most of devices correctly identified. However, the best configuration seems to fix $\gamma = 15$ and $\tau = 0.1$ with a slightly lower accuracy and a higher F-score. We will consider this configuration for the remaining experiments, otherwise it is mentioned. Table IV and V give the number of clusters and their sizes from this configuration to another by varying these two parameters. When τ increases, more trees are merged and so less larger clusters are built contrary to γ forcing trees to have more common neighbors for being linked when it increases. Some very small clusters are constructed with only one tree (outlier) since the minimum size is still zero. Furthermore, the original QROCK algorithm corresponding to $\gamma = 1$ is clearly unable to discover as many clusters as for $\gamma = 15$.

Figure 14 shows the evolution of the accuracy depending on the parameter ρ grouping the message arrived in the same interval of time. First, increasing ρ greater than five has no positive impact. Assuming a same device type for messages from the same IP address and port within 5 seconds seems reasonable. Second, the normalized distance ($dist_2$ for unsupervised fingerprinting) is better than the non-normalized one. It strengthen the usage of the normalized metric which does not need to be parameterized.

2) Message type: Only the most represented message types are considered : `100`, `200`, `401`, `OPTIONS`, `REGISTER`, `NOTIFY`, `INVITE` and `ACK`. The figure 15 plots the overall

γ (#neighbors)	τ (min distance) 0.01	0.05	0.1	0.15	0.2
1	0.559	0.767	0.721	0.307	0.302
5	0.480	0.748	801	0.306	0.306
10	0.454	0.742	0.872	0.307	0.307
15	0.424	0.727	0.862	0.525	0.307
20	0.370	0.698	0.804	0.524	0.307

$< 40\%$	40-60%	60-70%	70-80%	80-85%	$\geq 85\%$

TABLE II
UNSUPERVISED FINGERPRINTING BY GROUPING SIMILAR ARRIVAL TIME MESSAGES, DISTANCE $dist_2$ - ACCURACY

γ (#neighbors)	τ (min distance) 0.01	0.05	0.1	0.15	0.2
1	0.674	0.805	0.697	0.339	0.614
5	0.595	0.787	0.781	0.336	0.399
10	0.570	0.784	0.879	0.293	0.293
15	0.542	0.767	0.902	0.489	0.293
20	0.497	0.744	0.852	0.488	0.293

$< 40\%$	40-60%	60-70%	70-80%	80-85%	$\geq 85\%$

TABLE III
UNSUPERVISED FINGERPRINTING BY GROUPING SIMILAR ARRIVAL TIME MESSAGES, DISTANCE $dist_2$ - F-SCORE

accuracy and the F-Score of the classification results depending the type considered. Once again, best results are obtained with the normalized distance. Moreover, this graph points out that some types contain more valuable information than others. In order to reinforce the authentication mechanism, the `REGISTER` messages have to be considered. Despite the fact that accuracy is not high enough (70%) for discriminating intruders, imposing a stronger authentication to some users can rely on the fingerprinting results. Besides, `OPTIONS` message includes equipment capabilities which is highly dependent on the device type contrary to the response 200 which is only a kind of acknowledgment. However, this experiment shows that such messages also contain specific device information even if their semantic is not strongly related to the device type. Monitoring the right messages like OPTIONS or NOTIFY is very efficient because more than 90% of the messages are well classified with a similar F-score. Furthermore, this study is helpful for determining the strategy of packet sampling if needed. In our case (Intel Core 2 Duo 3GHz), comparing two syntactic trees requires 4.9ms and so selecting messages to classify can reduce the overall complexity.

τ	0.01	0.05	0.1	0.15	0.2
#clusters	314	108	61	33	14
Min size	1	1	1	1	1
Max size	126	218	222	480	720
Avg size	2.33	6.79	12.02	22.212	52.357

TABLE IV
CLUSTER CHARACTERISTICS WITH $\gamma = 15$

γ	1	5	10	15	20
#clusters	18	38	47	61	82
Min size	1	1	1	1	1
Max size	353	224	223	222	220
Avg size	40.72	19.29	15.60	12.02	8.94

TABLE V
CLUSTER CHARACTERISTICS WITH $\tau = 0.1$

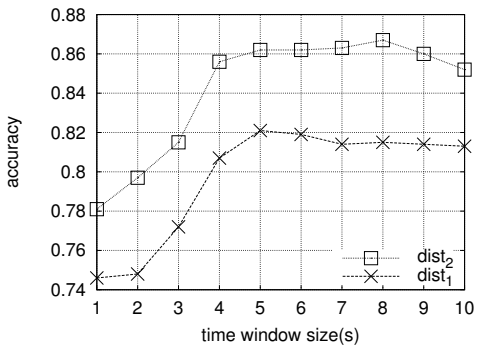


Fig. 14. Unsupervised fingerprinting by grouping similar arrival time messages

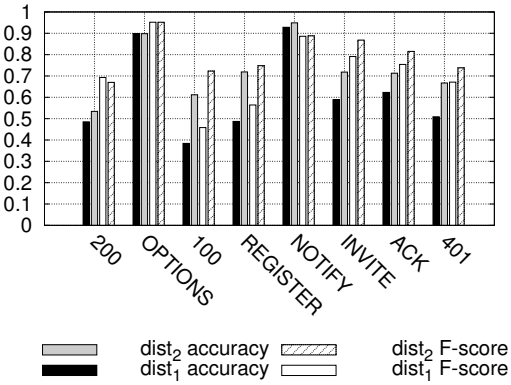


Fig. 15. Rock clustering by messages type

F. Real VoIP operator network dataset

This section addresses brief results obtained from the network traces of a real VoIP operator. This network contains 40 types of devices but the identification based on the SIP User-Agent returned sometimes an unknown type. Some types of devices are too much under-represented (less than 20 messages) while others generate 10.000 of the total of 96.000 messages from about 700 distinct devices. Hence, we discard four device types and keep at most 100 messages for each of them. The results are lower than for the testbed dataset. Indeed, the supervised fingerprinting technique is able to correctly identify 70% of the equipment. Assuming a classification per type of message, the unsupervised technique groups correctly 90% of OPTIONS message. The accuracy reaches 75% when messages within the same time interval are grouped beforehand. The first conclusion is that the OPTIONS message is a very valuable one. By investigating the reason of the relatively limited accuracy in the other cases, we found that some kind of devices cannot be well distinguished like

`CiscoATA186v3.1.0` and `CiscoATA186v3.1.1` due to small or no stack implementation modifications between the version 3.1.0 and 3.1.1. However, not detecting minor variations is not critical in some management applications as for example for general inventory purpose because the functionalities of such devices should be very similar. It may have a bigger impact for security applications. Furthermore, the correctness of this dataset could not be checked and the accuracy assessment is only based on the SIP User-Agent field which can be easily faked.

VII. Related work

Network and service fingerprinting tools are widely used by attackers for designing customized attacks or by network administrators to have a precise view of their network. The first work in this domain [19] highlights that unclear or permissive specification entails implementation variations due to specific choices or misunderstanding of the developers. Two classes of methods exist. The first one is qualified as passive since it only monitors the traffic without interacting with the fingerprinted machines. For instance, [20] is based on rules associating specific values in TCP fields to an operating system (OS). Active techniques send specific requests to a machine in order to get discriminative responses. This scheme is implemented by NMAP [21] for determining the OS. The accuracy of these techniques relies on the good definition of messages to send, which is basically done manually. Therefore, [22] describes a mechanism to learn them. Fingerprinting is not limited to OS identification and several works target to correctly distinguish the different kind of network traffic with different granularity levels. For instance, [23] gives a good overview of techniques used for determining the different classes of traffic (Web, P2P, Chat..) whereas [24] and [25] try to automatically identify a specific protocol. Our work is different and complementary since its goal is to determine precisely which implementation of a protocol is used. This kind of methods was explored in [26] for determining the HTTP [27] web server by observing the value or the order of some headers. Determining the version of a SIP equipment could be based on the bad randomness value of the Call-id field [15]. As argued in the introduction, changing these fields is very easy in order to counter fingerprinting. Our technique does not consider the value itself or the flat structure of the message but all its hierarchical syntactic structure related to the protocol grammar which contains more valuable information and which is more difficult to fake while keeping a valid message with the same meaning. SIP fingerprinting is also addressed in [28] with other protocol fields and an active probing technique contrary to the one presented in this paper which does not need any interaction with the devices. Our previous work [29] relies only on multiple sessions of messages without syntactic knowledge and is well designed for protocols with partially or unavailable specification and grammar. In addition, the fingerprinting process is longer because a minimal set of interactions is needed and the accuracy was lower with the same datasets (between 70 and 80%). We also introduced the use of syntactic information in [30] to create one generic global tree per device type. Even if the classification times are equivalent and it exhibits a very good accuracy (99% with 12% of messages used for training), the training process is very long and needs a grid of 10 computers during two days (with 2600 messages) contrary to the approach presented in this paper where the training process is very fast (few minutes). Thus, updating the system frequently is possible which is primordial with dynamic technologies like VoIP with many new devices appearing rapidly. Furthermore, our previous work did not deal with unsupervised fingerprinting. A syntax representation for applying support vectors machine techniques was also proposed in [31] by dividing a message into a pairs of attributed tokens. The objective there is different since the goal is to perform anomaly detection by detecting the deviation whereas we perform multi-class classification. Besides, the syntax structure proposed in [31] is composed of a sequence of attributed tokens whereas the approaches described in this paper rely on the syntactic tree. Dedicated kernel functions were proposed recently and [32] gives a good overview of the existing ones as for example the convolution kernels [33] and proposes other ones. A tree kernel function is basically defined by the number of subtrees shared between two trees with some variations. Our method can be regarded also as a variation since the isomorphism can be considered as shared subtrees. The main differences are that the nodes composing these subtrees are not to be exactly the same. We defined an equivalence function that considers sequence and repetition nodes to be semantically equivalent. Whereas usual tree kernel considers all shared subtrees, the method we propose needs only the subtrees rooted in the original root due to our specific context (section V-B).

VIII. Conclusion

This paper proposes novel supervised and unsupervised device fingerprinting techniques which leverage the advantages of the SVM paradigm and the ROCK classification. A new version of ROCK was introduced taking advantage of different pre-existing versions. The provided results show the viability of such fingerprinting schemes when used with syntactic trees which reflect both the content of messages and their hierarchical structures. Our future work will study the fingerprinting of other protocols with a focus on wireless protocols because their nature implies security problems like rogue machines intruding the network. We also plan to study the complexity of the approaches. Other directions include the automated monitoring of stack protocol implementation evolution of a device series.

References

[1] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, “RFC: Simple network management protocol (snmp),” United States, 1990.

[2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “RFC: SIP: Session Initiation Protocol,” United States, 2002.

[3] H. Abdelnur, T. Avanesov, M. Rusinowitch, and R. State, “Abusing SIP Authentication,” in *Information Assurance and Security (ISIAS)*. IEEE, 2008, pp. 237–242.

[4] L. Wang, Ed., *Support Vector Machines: Theory and Applications*, ser. Studies in Fuzziness and Soft Computing. Springer, 2005, vol. 177.

[5] L. Khan, M. Awad, and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering," *The VLDB Journal*, vol. 16, no. 4, pp. 507–521, 2007.

[6] M. Nassar, R. State, and O. Festor, "Monitoring SIP traffic using Support Vector Machines," in *11th International Symposium on Recent advances in intrusion detection - RAID 2008, Lecture Notes in Computer Science*, vol. 5230. Springer, 2008, pp. 311–330.

[7] R. Debnath, N. Takahide, and H. Takahashi, "A decision based one-against-one method for multi-class support vector machine," *Pattern Anal. Appl.*, vol. 7, no. 2, pp. 164–175, 2004.

[8] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415–425, Mar 2002.

[9] N. Cristianini and J. Shawe-Taylor, *An introduction to support Vector Machines: and other kernel-based learning methods*. New York, USA: Cambridge University Press, 2000.

[10] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping Multidimensional Data*, 2006, pp. 25–71.

[11] "Rock: A robust clustering algorithm for categorical attributes," in *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1999, p. 512.

[12] M. Dutta, A. K. Mahanta, and A. K. Pujari, "Qrock: a quick version of the rock algorithm for clustering of categorical data," *Pattern Recogn. Lett.*, vol. 26, no. 15, pp. 2364–2373, 2005.

[13] A. Torsello, D. Hidovic-Rowe, and M. Pelillo, "Polynomial-time metrics for attributed trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1087–1099, 2005.

[14] D. H. Crocker and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," United States, 1997.

[15] H. Scholz, "SIP Stack Fingerprinting and Stack Difference Attacks," *Black Hat Briefings*, 2006.

[16] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview." *Bioinformatics*, vol. 16, no. 5, pp. 412–24, 2000.

[17] C. J. van Rijsbergen, *Information Retrieval*. Butterworths, 1979.

[18] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, 1966.

[19] Douglas Comer and John C. Lin, "Probing TCP Implementations," in *USENIX Summer*, 1994, pp. 245–255. [Online]. Available: citeseer.ist.psu.edu/article/comer94probing.html

[20] "P0f," http://lcamtuf.coredump.cx/p0f.shtml.

[21] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. USA: Insecure, 2009.

[22] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum, "FiG: Automatic Fingerprint Generation," in *The 14th Annual Network & Distributed System Security Conference (NDSS 2007)*, February 2007.

[23] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the 2008 ACM CoNEXT Conference*. ACM, 2008, pp. 1–12.

[24] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference." in *Internet Measurement Conference*. ACM, 2006, pp. 313–326.

[25] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: automated construction of application signatures." in *MineNet*, S. Sen, C. Ji, D. Saha, and J. McCloskey, Eds. ACM, 2005, pp. 197–202.

[26] "Httpprint," http://www.net-square.com/httprint/httprint_paper.html.

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), June 1999, updated by RFC 2817. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[28] H. Yan, K. Sripanidkulchai, H. Zhang, Z. yin Shae, and D. Saha, "Incorporating Active Fingerprinting into SPIT Prevention Systems," *Third Annual VoIP Security Workshop*, June 2006.

[29] J. François, H. Abdelnur, R. State, and O. Festor, "Automated behavioral fingerprinting," in *12th International Symposium on Recent advances in intrusion detection - RAID 2009, Lecture Notes in Computer Science (To appear)*. Springer, 2009.

[30] H. Abdelnur, R. State, and O. Festor, "Advanced Network Fingerprinting," in *Recent Advances in Intrusion Detection Lecture Notes in Computer Science Computer Science*, ser. Computer Science, vol. Volume 5230/2008, MIT. Boston United States: Springer, 2008, pp. 372–389.

[31] P. Düssel, C. Gehl, P. Laskov, and K. Rieck, "Incorporation of application layer protocol syntax into anomaly detection," in *ICISS '08: Proceedings of the 4th International Conference on Information Systems Security*. Springer-Verlag, 2008, pp. 188–202.

[32] K. Rieck, "Machine learning for application-layer intrusion detection," Ph.D. dissertation, Fraunhofer Institute FIRST & Berlin Institute of Technology, 2009.

[33] M. Collins and N. Duffy, "Convolution kernels for natural language," in *Advances in Neural Information Processing Systems*. MIT Press, 2001.

Jérôme François is a research associate at the Interdisciplinary Centre for Security, Reliability and Trust of University of Luxembourg and was previously a Ph.D. research engineer at INRIA Nancy - Grand Est. He studied at ESIAL (Ecole Supérieure d'Informatique et ses Applications de Lorraine), a french leading school in computer science. In 2006, he obtained a Masters diploma in computer research. His Ph.D. twas supervised Olivier Festor and Radu State in the Madynes team in Loria laboratory (INRIA Lorraine - Nancy Grand Est France). He received his Ph.D. on robustness and identification of communicating applications from the Université Henry Poincaré in Nancy, France, in December 2009. Their current research activities are related to network security (fingerprinting and intrusion detection).

Humberto Abdelnur is a Ph.D research engineer at INRIA Nancy - Grand Est. He received his Ph.D on vulnerability assessment from the Université Henry Poincaré in Nancy, France, in 2009 and his MSc in Computer Science from National University of Cordoba (U.N.C.), Argentina, in 2005. His current research and interests are concerned to the fields of Software Reliability, Fuzz Testing and Network Fingerprinting.

Radu State holds a Ph.D from INRIA and a Master of Science in Engineering from the Johns Hopkins University (USA). He is a researcher in network security and network management with more than 60 papers published in international conferences and journals. He is member in the technical program committees of IEEE/IFIP Integrated Management, IEEE/IFIP Network Operations and Management and IEEE/IFIP DSOM, IEEE RAID. He lecturers at major conferences on topics related to security and network management and control. His activities range from network security assessment, software security to VoIP intrusion detection and assessment. He is currently on leave from INRIA and is associated with the University of Luxembourg.

Olivier Festor is a research director at INRIA Nancy Grand Est where he leads the MADYNES research team. He has a Ph.D. degree (1994) and an Habilitation degree (2001) from Henri-Poincare University, Nancy, France. He spent 3 years at the IBM European Networking Center in Heidelberg, Germany and one year at the EURECOM Institute in Nice, France. His research interests are in the design of algorithms and models for automated security management of large scale networks and services. This includes monitoring, fuzzing and vulnerability assessment. Application domains are IPv6, Voice over IP services and dynamic ad-hoc networks. He has published more than 70 papers in network and service management and serves in the technical program and organization committees as well as in the editorial boards of several international conferences and journals. He was the TPC Co-chair of the IFIP/IEEE IM 2005 event. Since 2006, he is leading the EMANICS European Network of Excellence dedicated to Management Solutions for the Future Internet and was named co-chair of the IFIP TC6 Working Group 6.6 co-chair in 2008.