HAL
open science

Behavioral and Temporal Fingerprinting

Jérôme François, Humberto Abdelnur, Radu State, Olivier Festor

To cite this version:

Jérôme François, Humberto Abdelnur, Radu State, Olivier Festor. Behavioral and Temporal Fingerprinting. [Intern report] RR-6995, INRIA. 2009, pp.22. inria-00406482v2

HAL Id: inria-00406482

https://inria.hal.science/inria-00406482v2

Submitted on 9 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Behavioral and Temporal Fingerprinting

Jérôme François — Humberto Abdelnur — Radu State — Olivier Festor

N° 6995

Juillet 2009

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6995--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND EST

Behavioral and Temporal Fingerprinting

Jérôme François*, Humberto Abdelnur† , Radu State‡ , Olivier Festor§

Thème : Remote device fingerprinting
Équipe-Projet Madynes

Rapport de recherche n° 6995 — Juillet 2009 — 22 pages

Abstract: This paper addresses the fingerprinting of communication protocols based on temporal and behavioral information. The objective of fingerprinting a device speaking a given protocol is to uniquely identify the device by looking at captured traffic that is generated by devices implementing that protocol. This paper proposes a conceptual model for capturing behavior and related temporal information from devices that implement a given protocol. Our key contribution is a fingerprinting scheme, where individual fingerprints are represented by tree-based temporal finite state machines. We have developed a fingerprinting scheme that leverages supervised learning approaches based on support vector machines for this purpose. We have validated the proposed approach on Session Initiation Protocol and concluded that very good classification performance is achieved.

Key-words: fingerprinting, support vector machines, finite-state machine

* jerome.francois@loria.fr
† humberto.abdelnur@loria.fr
‡ radu.state@loria.fr
§ olivier.festor@loria.fr

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Fingerprinting comportemental et temporel

Résumé : Ce document propose une nouvelle approche de fingerprinting des équipements grâce à l'analyse de leurs comportements en terme de messages échangés mais aussi grâce aux délais entre ces derniers. L'objectif est d'identifier chaque équipement différent utilisant le même protocole. Dans un premier temps, les informations comportementales et temporelles sont capturées et stockées dans des arbres dont la spécification formelle est définie dans ce document. Dans un second temps, une classification multi-classe supervisée basée sur les machines à vecteurs de support permet d'identifier les différents équipements. Cette nouvelle technique est validée grâce à de nombreuses expérimentations sur le protocole SIP dont les résultats sont très prometteurs.

Mots-clés : fingerprinting, signature, machines à vecteurs de support, machine à états finis

1 Introduction

Fingerprinting a device is an essential task in network security assessment, intrusion detection scenarios and copyright infringement trials. When performing a security assessment task, being able to identify the remote operating system and/or application is crucial for the accuracy and overall quality of the assessment. A similar case holds for intrusion detection and security monitoring. For instance, in a VoIP network that is operated only with a well-defined set of SIP (Session Initiation Protocol) [1] hardphone types, the presence of another type of end device could be detected, even if the overall security credentials are valid. Most application level protocols do contain information about the device identity (user agent) that generated the message, but in most cases it is not protected against malicious scrubbing. For instance, a popular and widely-deployed VoIP system used the following license agreements. A fixed number to tokens were distributed to the main VoIP server (call manager). A SIP device manufactured by a popular network vendor in the San Francisco area would consume one token, while any other SIP end device would consume at least two tokens. The device identity was asserted only from the user agent field (which was cryptographically unprotected) in the SIP signalling. We are convinced that an extended authentication mechanism can be built on top of device-level fingerprinting — for instance to allow only real hardphones to connect to a VoIP server.

In the past, several approaches for the remote identification of a device have been proposed. In most cases, these approaches were based either on identifying vendor- or device-specific deviations in the implementation of a given protocol. Such deviations are possible because of simple omissions in the specifications/norms — many current specifications either do not completely cover all the exceptional cases or lack the necessary precision, and thus leave to many degrees of freedom to the software implementers. The existing fingerprinting approaches exploit this in order to fingerprint a device. For instance, individual fields in the TCP header can lead to passive fingerprinting, as implemented in [2]. The main contribution of our paper is a new fingerprinting scheme that is accurate even in the case protocol stacks that are completely identical, but which are run on hardware having different capabilities (CPU power, memory resources, etc). We also look at the fingerprinting problem under more restrictive constraints. We propose a fingerprinting scheme that can learn distinctive patterns in the state machine of a particular implementation. We see such a pattern as a restricted tree finite state machine that provides additional time-related information about the transition performed. We define a similarity metric between patterns that is highly accurate for the classification of a given network capture.

Our paper is structured as follows: we start in section 2 with the presentation of the formal model that captures a temporal and state machine-related fingerprint. We continue in section 3 by explaining the functioning of our fingerprinting system and the way to construct the fingerprints. Section 4 is dedicated to the classification method. The metrics used for evaluating our system are described in section 5 and the datasets used for our experiments are detailed in the next section. Section 7 focuses on the determination of the best parameters based on a single dataset. Section 8 presents complete results from several datasets. Related work is summarized in section 9 before concluding.

2 Formal Model

We model a behavioral fingerprint using a Temporal Random Parameterized Tree Extended Finite State Machine (TR-FSM). The TR-FSM is an extension of the parameterized extended finite state machine introduced in [3]. Our extension concerns the introduction of temporal information and one additional constraint on the transitions in the state machine.

A TR-FSM is formally defined by a tuple
$M =< S, s_{init}, I, O, \vec{X}, T, \vec{Y} >$
where:

- S is a finite set of states with $|S| = n$;
- s_{init} is the initial state;
- $I = \{i_0(\vec{v_0}), i_1(\vec{v_1}), \ldots, i_{p-1}(\vec{i_{p-1}})\}$ is the input alphabet set of size p. Each symbol is associated with a vector of parameters;
- $O = \{o_0(\vec{w_0}), o_1(\vec{w_1}), \ldots, o_{q-1}(\vec{w_{q-1}})\}$ is the output alphabet set of size q. Each symbol is associated with a vector of parameters;
- $\vec{X}$ is a vector of variables;
- T is a finite set of transitions and each $t \in T$ is defined as $t =< s_1, s_2, i(\vec{v}), o(\vec{w}), P(\vec{X}, i(\vec{v})), Q(\vec{X}, i(\vec{v}), o(\vec{w})) >$. s_1 and s_2 are the start and end state, i is the input symbol triggering the transition and o is the triggered output symbol. $P(\vec{X}, i(\vec{v}))$ represents the condition to achieve the transition and $Q(\vec{X}, i(\vec{v}), o(\vec{w}))$ is the action triggered by the transition, based on an operation on the different parameters;
- $\vec{Y}$ is a $n-1$ dimensional random vector described later.

Additionally, the transitions of state machine are restricted to form a tree:

$$\forall s \in S \mid s \neq s_{init}, \exists\, !\ r \ states\ s_{i1}, s_{i2}, \ldots, s_{ir}$$

such that:

$$s_{i1} = s_{init} \wedge s_{ir} = s$$

where the notation ij represents a single index. The structure is a tree if there is only one possible sequence of transitions from the initial state to the destination state. Thus, we denote the corresponding transitions:

$$\forall j, 1 \leq j < r, t_{ij} \in T$$

$$t_{ij} =< s_{ij}, s_{i(j+1)}, i_{ij}(\vec{v_{ij}}), o(\vec{w}), P_{ij}(\vec{X}, i(\vec{v}))$$
$$Q_{ij}(\vec{X}, i_{ij}(\vec{v}), o_{ij}(\vec{w})) >$$

Hence, the cardinality of T is defined by $|T| = n-1$ and $T = \{t_1, \ldots, t_{n-1}\}$.

Finally $\vec{Y}$ is a $n-1$ dimensional random vector with Y_{t_j} representing the (measured) average time to perform the transition t_j.

In the rest of the paper, states and transitions are synonyms for nodes and edges because the TR-FSMs are trees and state machines also. Thus, a TR-FSM can be characterized by its **height** and its **cardinality** corresponding to $|S|$.

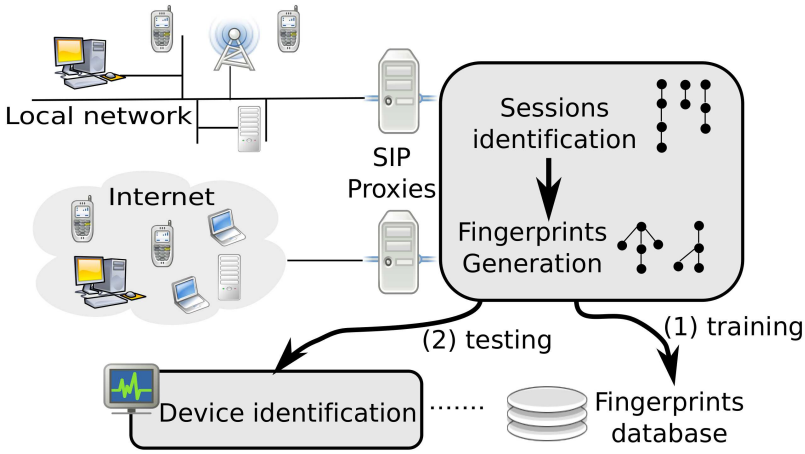


Figure 1: Fingerprinting architecture

One important note should be made. The location at which the measure is taken is important, especially when done from a remote site and over a network. The inherent additional noise due to the round-trip time can be filtered out. This is done by taking the network round-trip time into account. Alternatively, if the fingerprinting is integrated within an intrusion detection system, the measurements can be used directly without any other additional filtering, because in this case the system is learning local and deployment-specific parameterized device signatures.

The problem of fingerprinting can be now stated as follows. Given a candidate group of implementations $C = \{M_1, M_2, \ldots, M_k\}$ and a set of behavioral fingerprints $\{T_{j1}, T_{j2}, \ldots, T_{jp}\}$ for each implementation M_j, the goal is to find a classifier that correctly maps behavioral fingerprints to the corresponding classes.

We assume a similarity measure $\Delta(T_1, T_2)$, which is a distance based on the the tree structure and the vector $\vec{Y}$, between two TR-FSMs T_1 and T_2.

3 Fingerprinting framework

3.1 SIP background

We have considered SIP as a target application domain. We did this for several reasons. Firstly, SIP is widely deployed and the study of the security of VoIP currently is still in its infancy. Our goal was to provide a security monitoring framework that takes the fingerprint of a device into account. For instance, whenever one SIP device (identified by its IP address) shows a change in its underlying device fingerprint, this should be clearly treated as an attack. SIP messages are divided into two categories: requests and responses. Each request begins with a specific keyword like `REGISTER`, `INVITE`, `OPTIONS`, `UPDATE`, `NOTIFY`... The SIP responses begin with a three-digit numerical code

divided into six classes identified by the first digit. Figure 2(a) gives some examples of SIP sessions.

A **session** is composed of a sequence of messages and its delimitation depends on the protocol. Considering SIP protocol, a session is identified by a specific identifier (SIP call ID). Because an identifier can be reused several times, a session is considered finished after an inactivity period.

3.2 Architecture

Figure 1 depicts our fingerprinting architecture. First, the SIP traces are collected from the local network or Internet through a proxy where the clients are connected. Consequently, the clients are not connected via a dedicated network, entailing much noise on the traffic. The first step aims to identify the different sessions and to create the corresponding fingerprints as TR-FSMs (the next section details this step). The next stage is divided into two parts:

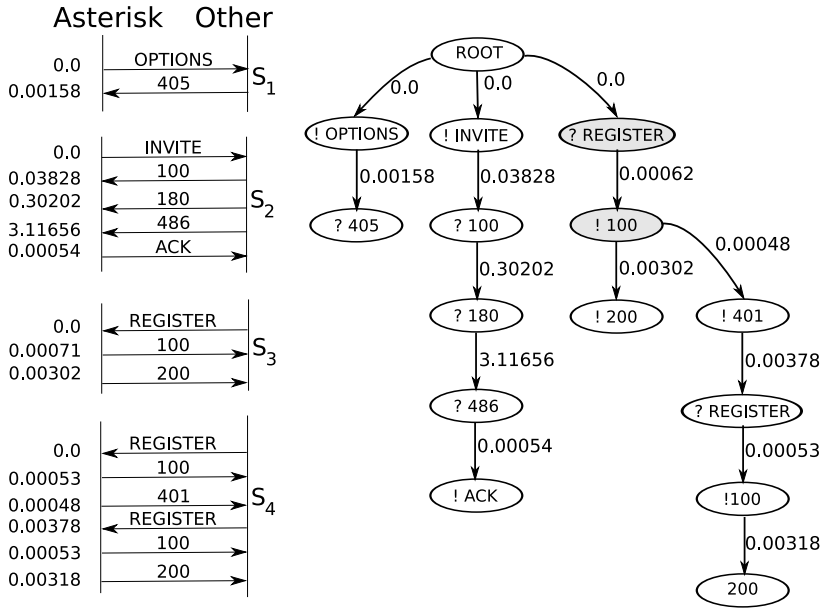
- during the learning phase (1), the fingerprints database is generated by identifying the devices using some knowledge of their characteristics. For example, the SIP user agent field (device identifier) can be used if the collected traces are assumed to be free of malformed messages.
- during the testing phase (2), the device identification module tests new fingerprints against the database in order to detect device changes or to check newly connected devices.

3.3 Fingerprint generation

The fingerprint is a tree with a generic ROOT node. The fingerprint represents a specific device and is generated from a subset of sessions in which this device participates. Each state of the TR-FSM is represented by a type composed of SIP request type or the SIP response code prefixed by **!** (outgoing message at the device fingerprinted) or **?** (ongoing message at the device fingerprinted). Figure 2(b) illustrates a TR-FSM corresponding to an Asterisk server. Therefore, nodes prefixed by **!** are messages sent by Asterisk, whereas those prefixed by **?** are emitted by any second party. This tree represents a signature for the Asterisk SIP proxy. A transition is indicated by an arrow between two states. In addition, the vector $\vec{Y}$ corresponds to the average delays put on the edges like in figure 2(b).

The signature in figure 2(b) is generated from the sessions shown in figure 2(a). In fact, each session is represented by a sequence of states and the shared prefixes are merged. For instance, the sessions S_3 and S_4 of the figure 2(a) have the two first messages in common and so they share the first two nodes which are gray colored in figure 2(b).

The algorithm 1 details the construction of a signature. For reasons of simplicity, the delay of a transition is directly stored on the node representing the end state without loss of information, since the tree structure involves only one ongoing edge for each node. Briefly, the algorithm maintains a current node initialized to the ROOT node. For each message m of the sessions, lines 16-18 aim to find a node n corresponding to the type of m among the children of the current node in order to update it. If this is not possible, a new node is



(a) Sessions (left value = time)

(b) A signature for Asterisk server generated from four sessions

Figure 2: Example of the fingerprint generation

created. The delay associated with an edge is the average delay in transmitting the corresponding message.

Considering a total of n messages, s sessions and the number of messages per session $n_i = |S_i|$, algorithm 1 iterates over all messages of all sessions, meaning that the number of iterations of lines 11 and 13 equals n. For each message, in the worst case the search (line 16) iterates over all possible children, which are at most as many as the previously examined sessions. Therefore the total number of iterations is $it = \sum_{i=1}^{s} i \times n_i$. Considering that all sessions except the last have only one message, we obtain the maximal value $it = s(n-(s-1)) + \sum_{i=1}^{s-1} i = ns + 1.5s - 0.5s^2 < ns + 1.5s$. Because, unlike n, the number of sessions to use is a fixed constant parameter, the overall complexity is $O(n)$.

4 Automated fingerprinting

4.1 Terminology

A dataset is composed of TR-FSMs. For a given dataset, the size N is the number of TR-FSMs $t_1, t_2, \ldots t_N$ that it contains. We follow the standard methodology in supervised learning. Each dataset is divided into a learning set used to train the system and a testing set. The testing set is used to evaluate the performance of the system when generalizing on new data. Each sub-dataset also has an associated size: N_train and N_test with $N = N_train + N_test$.

Algorithm 1 Tree construction

```
S a table containing the sessions
S_i^j is the node representation of the jth message of the ith session
tab.length returns the number of elements in tab
m.type returns the type of the message m prefixed by ? or ! depending on the direction
m.time returns the delay of the message m
n.children returns the child nodes of the node n
create_node(t) creates a new node from the message type t
n.update(d) updates the average delay of the ongoing edge of the node n using the delay d
n.add_child(n2, d) links the node n2 to n with an edge having delay d
n_ROOT is the root node
for i ← 1 to length(S) do
   current_node ← n_ROOT
   for j ← 1 to length(S_i) do
      child = current_node.children
      k ← 1
      while k < child.length ∧ child_k.type ≠ S_i^j.type do
         k ← ind + 1
      end while
      if k > child.length then
         new ← create_node(S_i^j.type)
         current_node.add_child(new, S_i^j.time)
         current_node ← new
      else
         child_k.update(S_i^j.time)
         current_node ← child_k
      end if
   end for
end for
```

The number of sessions extracted for building each tree is named **session size**: **training session size** for the training set and **test session-size** for testing set. These are important parameters for our method.

There are $N_devices$ distinct devices:
$D = d_1, d_2, \ldots d_{N_devices}$.

Two functions can be applied to each tree t_i:

- $real(t_i)$ returns the real identifier (device or implementation stack) for a TR-FSM t_i
- $assigned(t_i)$ returns the class name (device or implementation stack) for a TR-FSM t_i that is assigned by the fingerprinting scheme.

4.2 Supervised learning for fingerprinting

We briefly review the basics of support vector machines (SVM) in this section in order to make the paper self-contained. Additional reference material can be found in [4]. We relied on the multi-class classification [5] and adapted it to our fingerprinting task. The chosen approach is known as the one-to-one technique due to its good trade-off between classification accuracy and computational time [6].

Assuming the terminology of the previous section, the SVM classes correspond to the $N_devices$ devices, and the input space data points are the N_train trees from the training set. Firstly, each point t_i of the training set is

mapped to a high-dimensional feature space thanks a non-linear map function $\phi(t_i)$. Then, for each class pairwise $< c_l, c_k >$, an hyperplane with the maximum separation from both classes is found. First, we define the points involved for these classes:

$$\begin{aligned} T_l &= \{t_i | real(t_i) = c_l\} \\ T_k &= \{t_i | real(t_i) = c_k\} \end{aligned} \tag{1}$$

Then, the hyperplane is defined by a vector w^{lk} and a scalar b^{lk} and is under the following constraints:

$$\begin{aligned} \forall t_i \in \{T_l \cup T_k\} \\ \langle \phi(t_i) \cdot w^{lk} \rangle + b^{lk} \geq 1 - \xi^{lk}_{t_i},\ if\ real(t_i) = c_l \\ \langle \phi(t_i) \cdot w^{lk} \rangle + b^{lk} \geq -1 + \xi^{lk}_{t_i},\ if\ real(t_i) = c_k \end{aligned} \tag{2}$$

where the ξ terms are slack variables allowing some classification errors, *i.e.*, some points not on the correct side of the hyperplane because this is necessary when data points are not totally separable. The corresponding optimization problem can be converted to its dual form using the Lagrangian. Assuming that $\rho^{lk}_{t_i}$ is equal to 1 when $t_i \in T_L$ and -1 when $t_u \in T_K$, the problem is:

$$max \sum_{t_i \in \{T_l \cup T_k\}} \alpha^{lk}_{t_i} - \frac{1}{2} \sum_{\substack{t_i \in \{T_l \cup T_k\} \\ t_j \in \{T_l \cup T_k\}}} \alpha^{lk}_{t_i} \alpha^{lk}_{t_j} \rho^{lk}_{t_i} \rho^{lk}_{t_j} K(t_i, t_j) \tag{3}$$

subject to:

$$\begin{aligned} \sum_{t_i \in \{T_l \cup T_k\}} \alpha^{lk}_{t_i} \rho^{lk}_{t_i} = 0 \\ 0 \leq \alpha^{lk}_{t_i} \leq C,\ t_i \in \{T_l \cup T_k\} \end{aligned} \tag{4}$$

where K is a kernel function such as the following dot product:

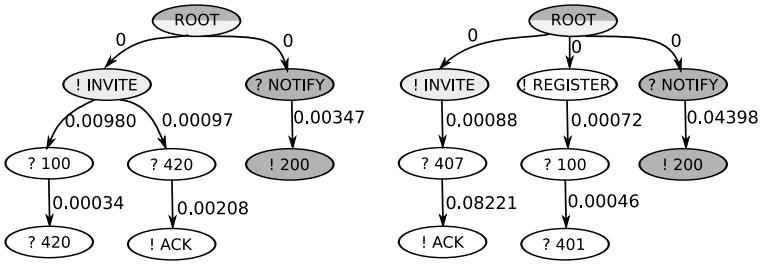
$$K(t_i, t_j) = \langle\ \phi(x_i).\phi_(x_j)\ \rangle \tag{5}$$

This kernel trick allows the problem to be solved without computing or knowing the ϕ function. The only requirement is a kernel function which has to be applied to each pair of data points. It is basically a similarity function constrained by Mercer's theorem [7]. Finally, a decision function, applied to each t_x of the testing set, is defined as:

$$f_{lk}(t_x) = \sum_{t_i \in \{T_l \cup T_k\}} \alpha^{lk}_{t_i} \rho_{t_i} K(t_i, t_x) + b^{lk} \tag{6}$$

In fact, the support vectors are the trees t_i with non-zero $\alpha^{lk}_{t_i}$ and form the set SV^{lk} from which b^{lk} is obtained:

$$b^{lk} = \frac{1}{|SV^{lk}|} \sum_{t_i \in SV^{lk}} (\rho^{lk}_{t_i} - \sum_{t_j \in \{T_l \cup T_k\}} \alpha^{lk}_{t_i} \rho^{lk}_{t_j} K(t_j, t_i)) \tag{7}$$



(a) Twinkle 1.10 (softphone)

(b) Cisco 7940 firmware 8.93 (hardphone)

Figure 3: Sessions tree examples of one hardphone and one softphone. The attribute on a directed edge is the average delay of the transition. Two shared paths are grey colored

During the testing stage, each decision function f_{lk} is applied to t_i, where t_i is a TR-FSM to classify. Depending on the return value, t_i is assigned to the class c_l or c_k. Using a voting scheme, the class chosen most often is considered to be correct.

Figure 3(b) shows a behavioral fingerprint for a SIP hardphone, while figure 3(a) presents a fingerprint for a softphone. However, the softphone makes one transition almost ten times faster then the hardphone. Therefore, if properly captured and used, time-related information can be be very useful when the same application is executed on different hardware, it will reflect differences in the architectural and computational features. For instance, the same SIP stack running on a CPU-limited capabilities hardphone will show higher transition times than the same stack on a high-performance workstation (softphone). The figures 3(b) and 3(a) illustrate this hypothesis.

4.3 Kernel function

The kernel function is one important parameter in SVM applications. The Gaussian kernel is a well-known possible function for simple data points given by a tuple of values. However, the current problem data points are trees with labelled edges. Therefore, we propose extending our previous method [8], based on the tree comparison method proposed in [9]. The goal is to obtain a similarity equal to 1 for exactly the same trees and 0 for totally different ones. Firstly, the set of paths from the root to each node of the tree t_i is designated by $paths^i$ and composed of m paths: $path^i_1, \ldots path^i_m$ where $path^i_j$ represents a single path. The function $nodes(path^i_j)$ returns only the nodes and transitions without delay properties. The function $nodes(paths^i)$ returns the set of the different paths $paths^i$ of the tree t_i without delays *i.e.*, the tree structure.

The intersection of the trees t_i and t_j is defined as:

$$I_{ij} = nodes(paths^i) \cap nodes(paths^j) \tag{8}$$

In figure 3, the two fingerprint intersections are shaded in gray.

For all shared paths, weight are derived from the delay differences and summed to obtain the similarity measure:

$$inter_sim = \sum_{\substack{p \in I_{ij} \\ nodes(path_k^i)=p \\ nodes(path_l^j)=p}} weight(paths_k^i, paths_l^j) \tag{9}$$

Without considering the delays, $path_l^j$ and $path_k^i$ are exactly the same for a given p. A comparison function is then calculated for each node $n_p \in p$ based on the Laplace kernel. Consequently, the new similarity measure is:

$$weight(p_1, p_2) = \sum_{n_p \in p_1} e^{-\alpha |f_{delay}(n,p_1) - f_{delay}(n,p_2)|} \tag{10}$$

where $f_{delay}(n, p)$ is a time-based function which returns the average delay for the ongoing edge from node n in the path p. Because a fingerprint concerns one device only, the delay due to other equipment has to be discarded, and so $f_{delay}(n, p) = 0$ for n a message received by the device (node name prefixed by ?).

Theorem *The following function is a valid kernel i.e., which satisfies Mercer's theorem (Chapter 3 of [7]):*

$$K(t_i, t_j) = \sum_{\substack{p \in I_{ij} \\ nodes(path_k^i)=p \\ nodes(path_l^j)=p}} \sum_{n_p \in p} e^{-\alpha |f_{delay}(n,p_1) - f_{delay}(n,p_2)|} \tag{11}$$

Proof: Eq. (10), which forms the inner sum, is a valid kernel known as Laplace Kernel K_l. The function $f_{delay}(n, p)$ can be expressed as a real-valued function $f(t_i)$ because n and p are subparts of t_i as well as t_j. Hence, the terms in the sum of K are expressed as $K_l(f(t_i), f(t_j))$, which is also a kernel due to kernel construction properties. Finally, a sum of kernels is also a kernel and so K is a kernel. Readers interested in kernel construction and related proofs are referred the section 3.3 in [7].

5 Performance evaluation

5.1 Metrics

Standard metrics for multi-class classification are defined in [10]. Obviously, the following functions are applied to testing trees only. The number of trees corresponding to a particular device d is denominated as x_d. The number of trees classified as device d is y_d. The number of trees classified as device d_1 and which correspond in reality to the device d_2 is $z_{d_2 d_1}$

The sensitivity of a device type d represents the percentage of the corresponding trees which are correctly identified:

$$sens(d) = z_{dd}/x_d \tag{12}$$

The specificity of a device d represents the percentage of trees which are labelled as d and which are really of this type.

$$spec(d) = z_{dd}/y_d \tag{13}$$

The overall metric, designated fingerprinting accuracy in this paper, corresponds to the percentage of trees correctly identified. The corresponding formula is:

$$acc = \sum_{d \in D} z_{dd}/N_test \tag{14}$$

The mutual information coefficient (IC) is a combination of entropies using the following distribution: $\mathbf{X} = x_i/N_test$, $\mathbf{Y} = y_i/N_test$, $\mathbf{Z} = z_{ij}/N_test$. It is defined as:

$$IC = \frac{H(\mathbf{X}) + H(\mathbf{Y}) - H(\mathbf{Z})}{H(\mathbf{X})} \tag{15}$$

where H is the entropy function. This IC is a ratio between 0 and 1 and is maximal for a perfect classification. It is very useful to compare classifications with the same overall accuracy. In this case, the ratio can be degraded when each class is not well represented. For example, it is easy to obtain an accuracy of 80% if 80% of data points are of the same type by assigning all of them to a single class. However, in this case the information coefficient will be 0.

6 Experimental datasets

We made extensive use of network traces from which we could extract the SIP user agent (device type) in order to perform both the training and the testing our system. We assumed that our traces did not contain malicious messages, where for instance an attacker spoofed the user agent field. Our implementation is based on the LIBSVM library [11].

We used two kinds of datasets. The first was generated from our testbed composed of various end-user equipment including softphones like `Twinkle` or `Ekiga` and hardphones from the following brands: Cisco, Linksys, Snom or Thomson. The testbed also used servers such as `Asterisk` and OpenSer/Cisco Call Manager. This dataset will be described as `testbed dataset` in the remainder of the paper. The other datasets designated `operator datasets` (T1 to T4) were provided by four real VoIP operators (about 45MB of traces were extracted). Most equipment is hardphones or SIP servers. The main difference between the two kinds of dataset is the network environment. The first characterizes a local network, while the `operator datasets` capture traffic from devices that connect from the Internet. This implies greater noise and longer delays, as shown in the table 1. We used these different target environments intentionally in order to validate the robustness of our approach in noisy conditions. Obviously, the time delays are relevant when comparing different datasets, but within one dataset, the fingerprinting process should be able to properly identify each device. Table 1 shows main characteristics of the datasets: the number of different devices, the number of messages, as well as the number of INVITE messages, which indicates the number of VoIP calls made through the network. Although the `operator datasets` are more complete in terms of messages and devices, the number of INVITEs is quite low, indicating that most of the SIP sessions

	Testbed	T1	T2	T3	T4
#devices	26	40	42	40	40
#messages	18066	96033	95908	96073	96031
#INVITE	3183	1861	1666	1464	1528
#sessions	2686	30006	29775	30328	30063
Avg #msgs/session	6.73	3.20	3.22	3.16	3.20
Avg delay (sec)	1.53	7.32	6.76	6.11	8.52

Table 1: Experimental datasets statistics

are not phone calls, but registration requests. This reflects realistic SIP traffic, as all SIP user agents have to periodically send out a registration request in order to maintain the binding between a SIP AOR (the generic and global identifier for a user) and the current IP address. Being able to fingerprint devices just by looking at the registration messages is also important for device level authentication.

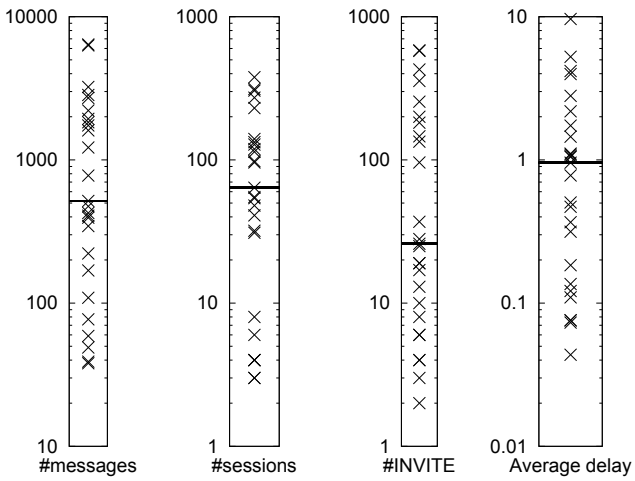
Figure 4 highlights some of the differences between the devices for the `testbed dataset` and the first operator T1. Each point in the figure represents one device. We considered only messages emitted by the corresponding device and we used a logarithmic scale. For the two datasets, the distribution of messages per device is obviously not uniform, reflecting reality because some devices are used more than others. Thus, this implies that the differences between devices for the number of sessions and INVITE messages are similar. Additionally, the distribution ranges of the number of messages and the number of sessions is greater for the operator T1 (figure 4(b)). Hence, the differences between devices are highlighted. For instance, one device has only generated one SIP session while another more than 10,000 as shown on the second graph of figure 4(b).

Due to the difference in the average time delays, it seems possible to fingerprint devices based on such pieces of information. However, when these differences are however insignificant, additional information is needed. Our approach combines the temporal aspect with the behavioral aspect. For example, in figure 4(b), four or five groups of devices can be easily identified just by comparing the average delays. Considering the dataset T1, the transition delays are generally higher than for `testbed dataset` and the median value is doubled. Moreover, many devices have not sent any INVITE message and so are not plotted on the graph due to the logarithmic scale. The median value is also zero and so not plotted.

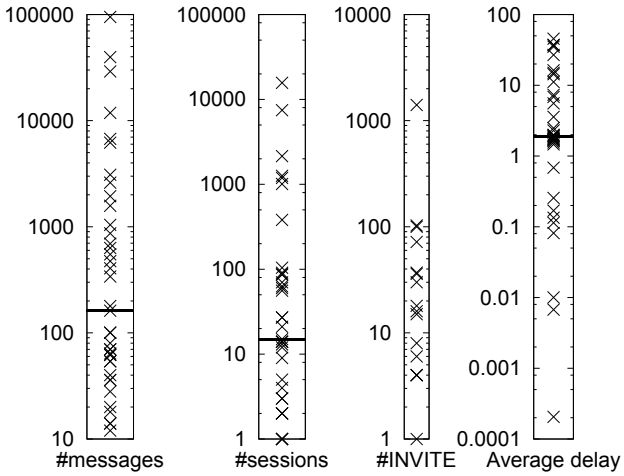
7 `testbed dataset` results

The characteristics of the `testbed dataset` are depicted in figure 4(a) and in table 1. We used it to assess the accuracy of the behavioral and temporal fingerprinting. One objective was to determine the impact of the different parameters on these performance metrics and tune them. These tuned parameters would then be used on the larger `operator datasets`.

We randomly selected 40% of the sessions of each device to form the training set. The remainder (60%) represents the testing set. Each experiments was run



(a) Testbed



(b) Operator T1

Figure 4: Experimental dataset statistics by device (Logarithmic scale; horizontal black bar is the median value; each point represents a device)

ten times, shuffling the sessions before selection in order to improve the validity of the experiments. Then, the average values over the different instances of the classification metric are considered. Furthermore, we use quartiles to gain an idea of the distribution of the results. Figure 5 represents quartiles, where the extrema are the minimal and maximal observed values. The lower limit of the box indicates that 25% of the observations are below this value. The upper limit of the box is interpreted in the same way with a percentage of 75%. Finally the horizontal line inside the box is the median value. Therefore, the box contains the 50% of the observations closest to the median.

With the exception of Section 7.3, α_1 and α_2 are set to 1000.

7.1 Session-size tree

We first investigate the optimal session sizes for training. The test session-size is more important because it shows how reactive the system is. In the best case, a session size of one implies the recognition of one device with only one session. Secondly, we look at the relationship between testing session size and training session size.

Table 2 provides a short summary of this data. The shading key simply highlights the main observations concerning fingerprinting accuracy. Our technique cannot be applied to detect a device with only one session (first column is very pale). The darkest row corresponds to a train session-size of five. The training process does not need both huge trees and many sessions because the greater the session size is, the more necessary the sessions. Using a training session size of five and a testing session size of ten, the maximal accuracy ($\sim$ 90%) is obtained. Subsequent experiments assume this optimal configuration. It can be seen that, even if our technique is not designed for single session device identification, its results are very good. Using only ten sessions or even five sessions, the corresponding accuracy is about 86%.

Finally, the low standard deviation shown in brackets indicates that the accuracy is still about the same during the different experiments especially in the best configrations (dark gray).

Regarding the average sensitivity appearing in table 3, the optimal configuration is still the same and the corresponding accuracy is 65%. This relatively low result is due mainly to some incorrectly fingerprinted devices. In fact, some devices are poorly represented in the dataset as shown in figure 4(a). For instance, a training session size of five and a training set of 40% of sessions results in a minimal number of $\lceil 5/0.4 \rceil = 13$, sessions which is not the case for six devices (figure 4(a). Furthermore, this minimal value implies only one training tree and all learning clustering techniques need more training data for efficiency. The impact of training set size is studied in the next subsection.

Although comparing identically-sized trees seems more logical and probably more efficient, this experiment shows the reverse due primary to our comparison function, which considers the various paths in the trees separately (see equations (8)-(11)).

7.2 Training set size

As it was previously mentioned, the fingerprinting accuracy per device is much affected by underrepresented devices. We assess the minimal training trees per device capable of achieving good results. This number varies from 1 to 20 in figure 5. Firstly, if there are at least two trees for each device, the accuracy is more than 80% in most cases. Thus, a training session size of 5 implies at least $5 \times 2 = 10$ sessions for the training process, which is reasonable. Going further, the accuracy is close to 90% for a minimal training set size equals eight.

7.3 Effect of the α parameter

The parameter α is introduced in formula (11), and has a potential impact on fingerprinting accuracy, since it impacts the average delay weight. The higher α is, the more important are small delay differences. Figure 6 highlights the impact

Training session size	Testing session size				
	1	5	10	20	40
1	0.682 (0.009)	0.819 (0.013)	0.830 (0.013)	0.805 (0.031)	0.745 (0.034)
5	0.469 (0.028)	0.858 (0.013)	0.905 (0.011)	0.883 (0.025)	0.800 (0.035)
10	0.376 (0.044)	0.809 (0.011)	0.894 (0.013)	0.873 (0.021)	0.819 (0.035)
20	0.272 (0.028)	0.656 (0.028)	0.821 (0.015)	0.864 (0.015)	0.837 (0.012)
40	0.221 (0.027)	0.469 (0.026)	0.627 (0.030)	0.764 (0.037)	0.762 (0.038)

$< 50\%$	50-70%	70-80%	80-85%	85-90%	$\geq 90\%$

Table 2: `testbed dataset`: Average fingerprinting accuracy (standard deviation is put in brackets)

Training session size	Testing session size				
	1	5	10	20	40
1	0.504 (0.011)	0.542 (0.034)	0.553 (0.032)	0.535 (0.044)	0.529 (0.043)
5	0.294 (0.026)	0.605 (0.035)	0.647 (0.035)	0.648 (0.047)	0.580 (0.045)
10	0.224 (0.028)	0.550 (0.017)	0.625 (0.023)	0.636 (0.024)	0.599 (0.047)
20	0.145 (0.021)	0.452 (0.050)	0.572 (0.030)	0.615 (0.045)	0.622 (0.027)
40	0.109 (0.028)	0.316 (0.030)	0.399 (0.032)	0.505 (0.050)	0.522 (0.038)

$< 30\%$	30-40%	40-50%	50-55%	55-60%	$\geq 60\%$

Table 3: `testbed dataset`: Average sensitivity (standard deviation is put in brackets)

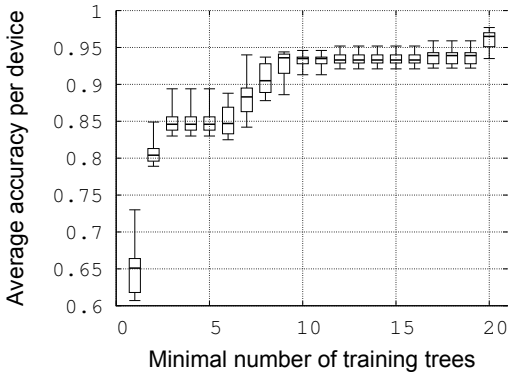


Figure 5: `testbed dataset`: Learning trees minimal number impact (test session-size = 10, training session size = 5, $\alpha = 1000$)

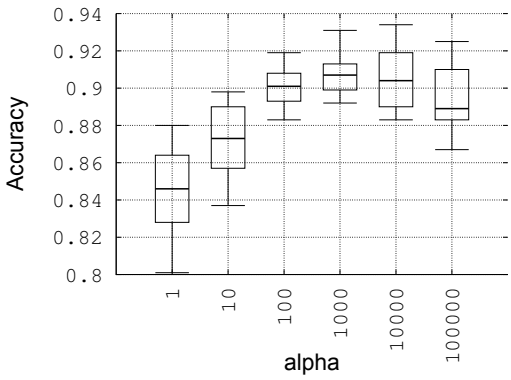


Figure 6: `testbed dataset`: α parameter impact (testing session size = 10, training session size = 5)

of α on average accuracy by showing the quartiles. Its shape is a parabola with smallest values at the extremities. Broadly, when considering a reference time, a difference between 1 second and 4 second has to be interpreted differently from the difference between 56 and 59 seconds. This can be achieved by increasing α. However, when α is too high, the difference between 0.1 second and 0.2 second could be too discriminatory. This means that the correct trade-off is the maximal value on figure 6. In fact, the values 100, 1000 or 10000 are possible choices because accuracy is similar. However, we prefer $\alpha = 1000$ as the median value is the best, and above all the results are concentrated very close: around the median. We expect a similar accuracy independent of the sessions selected for use in the training stage.

8 Global results

We will consider a train session-size of five and a test-session of ten because this configuration previously gave the best results. Table 2 gives all statistics and results. The initial rows are related to dataset statistics. Considering the `testbed dataset`, even when more sessions are selected for the testing process, the number of testing trees is lower due to a higher test session-size. Each experiment is performed three times for the `operator datasets` and ten times for the `testbed dataset`. Except for the number of trees, which is fixed for all experiments, the average values are given, with the standard deviation in brackets. For the `operator datasets`, only 10% of sessions are used for the training stage. It is important to note that the standard deviation of maximal and average heights and cardinality is high. This shows that our experiments cover many configurations. At the same time, the classification results in the lower part of the table are stable, as highlighted by a low standard deviation, demonstrating that our fingerprinting approach is suited to many distinct configurations. Obviously, the TR-FSMs of the `operator datasets` are higher and bigger because the datasets are more complete.

Considering the operators, the overall accuracy reaches about 86%, which is lower than the `testbed dataset` (91%), due principally to additional noise on Internet. Moreover, the mutual information coefficient (IC) for the `testbed dataset` is very high, indicating that the high accuracy is not due an over-represented device. However, this coefficient is lower for the `operator datasets` because some devices are clearly present ingreater numbers than others, as highlighted in 4(b). Once again, for several devices, the number of sessions is too low to have complete training sets and so the average sensitivity is concentrated between 45% and 58%. However, the specificity is always high, meaning that the misclassified trees are well-scattered among the different devices.

9 Related work

Network and service fingerprinting is a common task that is often used by attackers to design efficient attacks. However, it is also a useful and legitimate tool for security assessment, penetration testing and monitoring the diversity of hardware and software on the network. The key assumption is that subtle differences due to development choices and/or incomplete specification can be traced back the specific device/protocol stack [12]. There are two main classes of fingerprinting scheme: active and passive. Passive fingerprinting monitors network traffic without any interaction. The most efficient tool for this purpose is p0f [2], which uses a set of signatures to identify the operating system that generated a TCP packet. Each signature is based on the specific values in particular TCP/IP header fields. In contrast, active fingerprinting generates specific requests directed to a device and monitors the responses. For instance, [13] implements this scheme in order to detect the operating system and service versioning of a remote device. A related work is [14], which describes active probing and proposes a mechanism to automatically explore and select the right requests to make. These requests can themselves considered as fingerprints themselves.

Metric	Testbed	T1	T2	T3	T4
#Training trees	440	1223	1217	1237	1224
#Testing trees	332	5409	5367	5471	5423
Max height	71.95 (32.03)	464.67 (41.35)	476.33 (38.58)	420.33 (30.56)	431.33 (0.94)
Min height	1.9 (0.30)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Avg height	9.53 (2.13)	8.80 (1.53)	8.85 (1.89)	8.70 (1.73)	9.05 (1.38)
Max card	89.00 (35.72)	492.67 (44.68)	491.17 (47.65)	540.84 (157.00)	464.84 (21.52)
Min card	3.95 (1.56)	2.67 (0.47)	2.00 (0.00)	2.00 (0.00)	3.00 (0.00)
Avg card	18.97 (4.69)	12.93 (2.68)	12.94 (3.09)	12.85 (2.98)	13.23 (2.56)
Accuracy	0.91 (0.011)	0.81 (0.004)	0.86 (0.001)	0.85 (0.002)	0.83 (0.004)
Sensitivity	0.64 (0.030)	0.53 (0.019)	0.58 (0.026)	0.54 (0.012)	0.43 (0.015)
Specificity	0.91 (0.035)	0.79 (0.001)	0.81 (0.025)	0.77 (0.028)	0.77 (0.028)
IC	0.87 (0.012)	0.64 (0.001)	0.65 (0.001)	0.65 (0.003)	0.63 (0.004)

Table 4: Experimental datasets results ($\alpha = 1000$, test session-size = 10, train session-size = 5). Average values given and standard deviations in brackets

Fingerprinting might have also another interpretations: for instance [15] and [16] focus on the identification on the flow types, unlike our method which aims to distinguish specific implementation of a protocol.

The fingerprinting of SIP devices has already been addressed in [17, 18, 8]. However all these works consider that it is possible to extract specific SIP fields. We have addressed a somewhat related topic in [19], where we looked at the identification of the different message types used by an unknown protocol and were able to build up the tracking state machines from network traces. That approach can serve to build TR-FSMs for an unknown protocol without any domain-specific knowledge, especially of the grammar of the protocol. Secondly, we have not until now considered both behavioral and temporal aspects of the fingerprinting task at the same time. Support vector machines have been already proposed in the context of network security monitoring and intrusion detection. For instance, [20] addresses the issue of SVMs in intrusion detection approaches, while our own previous work [21] showed a VoIP-specific application for SVMs. None of the previous related work addressed the construction of time based behavioral fingerprints.

Construction of the state machine of a protocol from a set of examples has been studied in the past. Although known to be NP complete (see [22],[23] and [24] for good overviews on this topic), the existing heuristics for this task it are based on building tree representations for the underlying finite state machine. In our approach we do not prune the tree and, although the final tree representation is dependent on the order in which we constructed the tree, we argue that the resulting substrees have good discriminative features. We developed a classification method based on tree kernels in order to take into account the peculiar nature of the input space. Tree kernels for support vector machines have recently been introduced in [25], [26], [27] and allow the use of substructures of the original sets as features. Our approach extends this concept in order to be applicable to the TR-FSMs we defined. In consequence, a new valid kernel is proposed in this paper.

10 Conclusion

In this paper, we have addressed the problem of fingerprinting devices and/or implementation stacks. Our approach is based on the analysis of temporal and state-machine-induced features. We introduced the TR-FSM, a tree-structured parameterized finite state machine having time-annotated edges. A TR-FSM represents a fingerprint for device/stack. Several such fingerprints are associated with a device. We propose a supervised learning method, where support vector machines use kernel functions defined over the space of TR-FSMs. We validated our approach using SIP as a target protocol. We will continue this work in two main directions. Firstly, we will look at other protocols — for instance wireless protocols — and assess the operational applicability in this scenario. This would for instance allow the identification of rogue access points within a large wireless access infrastructure. A second research direction consists of defining other kernel functions specific to the TR-FSMs that allow the modeling of the probability distribution of transition times at each edge. This will leverage not only the average transition time for one edge, but also the underlying probability distribution.

References

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

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

[3] G. Shu and D. Lee, "Network protocol system fingerprinting - a formal approach," in *25th IEEE International Conference on Computer Communications, INFOCOM*, 2006.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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, Ari Trachtenberg, Ed., vol. Volume 5230/2008, MIT. Boston United States: Springer, 2008, pp. 372–389.

[9] D. Buttler, "A Short Survey of Document Structure Similarity Algorithms," in *The 5th International Conference on Internet Computing*, Jun. 2005.

[10] 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.

[11] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[12] 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

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

[14] 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.

[15] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference." in *Internet Measurement Conference*, J. M. Almeida, V. A. F. Almeida, and P. Barford, Eds. ACM, 2006, pp. 313–326.

[16] 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.

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

[18] 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.

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

[20] 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.

[21] 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 Recent Advances in Intrusion Detection Lecture Notes in Computer Science*, vol. 5230. Springer, 2008, pp. 311–330.

[22] R. L. Rivest and R. E. Schapire, "Inference of finite automata using homing sequences," in *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1989, pp. 411–420.

[23] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[24] R. E. Schapire, "Diversity-based inference of finite automata," Cambridge, MA, USA, Tech. Rep., 1988.

[25] M. Collins and N. Duffy, "New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron," in *ACL02*, 2002.

[26] S. Vishwanathan and A. Smola, "Fast kernels on strings and trees." in *Proceedings of Neural Information Processing Systems*, 2002.

[27] A. Moschitti, "Making tree kernels practical for natural language learning," in *Proceedings of the Eleventh International Conference on European Association for Computational Linguistics*, 2006.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399