Continuous and Non-Intrusive Reauthentication of Web Sessions based on Mouse Dynamics

Eric Medvet, Alberto Bartoli, Francesca Boem, Fabiano Tarlao
Department of Engineering and Architecture
University of Trieste, Italy
emedvet@units.it, bartoli.alberto@units.it, fboem@units.it, fabiano.tarlao@phd.units.it

***Abstract*—We propose a system for continuous reauthentication of web users based on the observed mouse dynamics. Key feature of our proposal is that no specific software needs to be installed on client machines, which allows to easily integrate continuous reauthentication capabilities into the existing infrastructure of large organizations. We assess our proposal with real data from 24 users, collected during normal working activity for several working days. We obtain accuracy in the order of 97%, which is aligned with earlier proposals requiring instrumentation of client workstations for intercepting all mouse activity—quite a strong requirement for large organizations. Our proposal may constitute an effective layer for a defense-in-depth strategy in several key scenarios: web applications hosted in the cloud, where users authenticate with standard mechanisms; organizations which allow local users to access external web applications, and enterprise applications hosted in local servers or private cloud facilities.**

***Keywords*-behavioral biometric; defense-in-depth**

I. Introduction

Stealing of authentication credentials has become a first class security problem which cannot be considered an exceptional event [1], [2], [3], [4]. Mechanisms capable of complementing the traditional authentication procedures, which are based on knowledge of a certain password or possession of a certain cryptographic key, would be highly desirable. Approaches which have been proposed in this respect include usage of the stream of events generated at the human-machine boundary as a "behavioral biometric" property which can be univocally associated with each user—keystrokes [5], [6], mouse trajectories and clicks [7], [8], [9], [10], [11], [12], [6], touch-screen interactions [13]. By comparing the stream of events in a certain session to a previously collected ground truth, one may perform additional authentication checks with high accuracy. Approaches of this kind usually assume that the client machine is instrumented with software capable of collecting all the user-generated events of interest, either at login time or continuously in the background. We believe that this requirement may be too difficult to satisfy in practice, especially for large organizations, and in this work we investigate the feasibility of an alternative approach which is much simpler to implement and deploy.

We consider the problem of using mouse dynamics for *continuous reauthentication* in a setting where no specific software may be installed on client machines and mouse-generated events are available only for web traffic. The model resulting from these assumptions allows integrating continuous reauthentication capabilities into the existing infrastructure of large organizations easily. In particular, our model fits several key scenarios: web applications hosted in the cloud, where users authenticate with standard mechanisms; organizations which allow locally authenticated users to access external web applications, and enterprise applications hosted in local servers or private cloud facilities.

The problem of identifying users by means of biometric data comes in two flavors [14]: *verification*, in which the system is required to check whether the current user is really the user he/she claims to be; and *identification*, in which the system is required to identify which is the current user amongst a population of known users. In principle, mouse dynamics might be used for both verification and identification. In this work, we consider a form of verification, because we assume that the connected user claims a certain identity by successfully executing some authentication procedure (the specifics of this procedure are irrelevant). The task of the system consists in continuously checking the actual mouse dynamics and generating an alert in case the observed data do not fit the mouse dynamics of the claimed user. In other words, we do not advocate usage of mouse dynamics as the only tool for authenticating users and suggest instead its use as a layer for a defense-in-depth strategy, i.e., as a complement to other forms of authentication, intrusion detection, and so on. In this respect, mouse dynamics fit neatly into an emerging framework where authentication credentials are considered just one of the multiple signals to be used for authenticating humans [15]. The threat model assumes an attacker who impersonates a legitimate user in web browsing sessions which last for several minutes on a mouse-equipped platform. This model fits, in particular, credential stealing scenarios where an attacker occasionally or routinely accesses an account fraudulently. The model does not address attackers who perform a session lasting just a few seconds (more details in Section III-C).

Our contribution is the following: (a) we describe a system for capturing GUI-related events for web traffic which does not require any specific software to be installed on client machines and is fully *transparent* to both users and web sites; (b) we describe a procedure for performing continuous reauthentication (i.e., frequent verification of the claimed user identity) based on the observed mouse-generated events; and, (c) we show, based on real data collected in two distinct working environments, that de-

978-1-4799-4223-7/14 $31.00 © 2014 IEEE
DOI 10.1109/ARES.2014.29

spite the intrinsic limitations of the collection procedure, with respect to the commonly adopted approach of instrumenting client machines for collecting all user activities, the system exhibits accuracy aligned with the state-of-the-art.

Our system consists of an HTTP proxy which may be deployed either close to servers or close to clients, and of a specialized *Collector* application. The proxy is configured for injecting JavaScript code which captures mouse-generated events into all web documents fetched through the proxy. The JavaScript code then sends the collected events in batches to the Collector, which performs continuous reauthentication in the background by comparing the observed mouse dynamics to the mouse dynamics of the claimed user. The proxy also rewrites the fetched web documents in order to circumvent the Same Origin Policy enforced by browsers [16], which would prevent the transmission of data to Collector while interacting with pages fetched from a different web site.

We construct mouse dynamics in a feature space similar to several earlier proposals (e.g., [7], [10]) and composed of 39 features related to position, speed, acceleration, and so on. In order to minimize the amount of events to be collected and sent to the Collector, we chose to: (a) construct a new feature vector only when there is a pause of at least 500 ms in mouse usage; and (b) use in each feature vector only a small amount of the events immediately preceding each pause. As it turns out from our experimental evaluation, this design choice does not harm detection accuracy. We execute continuous reauthentication whenever there is a new feature vector available, by applying a Support Vector Machine calibrated specifically for the user who claims to be connected. We generate an alert whenever the number of recent feature vectors which do not fit the expected profile exceeds a certain user-dependent threshold. This simple filtering boosts the accuracy which one would obtain by considering only the last feature vector generated, while at the same time keeping the time required for generating the first alert in the order of the tens of minutes—a reaction time which, in our opinion, is both reasonable and realistic for a reauthentication mechanism of this kind applied to the considered threat model. We remark that the choice of the threshold is based on training data only and is an integral part of our calibration methodology, that is, we do not focus our analysis on the threshold values which happen to provide the best performance on testing data.

We assess our proposal with real data from 24 users, collected during normal working activity for several working days. We obtain accuracy in the order of 97%, which is aligned with earlier proposals requiring instrumentation of client workstations for intercepting all mouse activity.

II. Data capture system

Our data capture system is fully *transparent* to both the user and the target web sites. That is, the user navigates with a normal browser and the target web sites do not need to be modified in any way. The system consists of: (a) a web application which we developed and that we call *Collector*; and (b) a *proxy* which must be placed in between the browser and the target web sites. The system captures all mouse-related events generated by web traffic which travels through the proxy. Each user may thus use his/her preferred browser. The two components need not be physically separated, that is, an organization might choose to integrate Collector in the proxy.

The Collector is composed of a server side code (Collector-S) and a client-side code executed by the browser (Collector-C). Collector-C records all the mouse-related DOM events generated by the user and periodically sends a description of these events to Collector-S, which performs continuous reauthentication in the background by comparing the observed mouse dynamics to the mouse dynamics of the claimed user and alerts administrators in case of a mismatch. Collector-C is able to record also keyboard-related events, which may potentially improve the quality of continuous reauthentication even further, but we have not exploited this possibility in this work. We developed Collector with the Google Web Toolkit (GWT). GWT is a Java framework which allows writing AJAX web applications entirely in Java and is able to transcompile Java code directly in JavaScript code ready to be executed by any web browser.

The proxy: (i) injects the Collector-C code into all the pages sent to the browser, to make it possible recording user's actions transparently to the target site; and (ii) redirects part of the web traffic so as to enable communication between Collector-C and Collector-S without violating the *same origin policy* (SOP) [16] implemented by modern browsers (see below). The code injected by the proxy takes this form: `<script type="text/javascript" src="/GWT-Observer/observer.js"></script>` Upon parsing the received page, the browser will fetch the JavaScript code—i.e., the Collector-C code—from the specified `src` location and execute that code locally. The results produced by Collector-C are sent to `/GWT-Observer`. Since both the `src` location and the `/GWT-Observer` are specified by means of a relative URL, the browser treats our code as if it was part of the monitored web application. In other words, the browser is tricked into believing that Collector-C is fetched from the target web site and communicates with that site. The proxy is configured so as to reroute any traffic to `/GWT-Observer` toward the server in our control which actually executes the Collector-S code.

The Collector-C code is obfuscated and its size is approximately 70 kB, which drops to approximately 22 kB if the browser accepts compressed content. Upload bandwidth usage is in the order of $2.5\,\mathrm{kB\,s^{-1}}$. Although we have not performed a systematic performance analysis, we have not experienced any observable performance penalty during continued usage of the system for our normal working activity. This is not surprising having considered that navigation in modern web sites involves loading many thousands of JavaScript functions whose aggregate size is

in the order of the MegaBytes [17]. Indeed, an indirect proof that our approach does not hurt performance is that several large providers are already recording mouse-generated events of their users [18].

The system is highly flexible and may be deployed in a variety of ways. It could be deployed at the organization hosting the web application or at the boundary of an organization for monitoring all outbound web traffic. In the former case the system would monitor a web application, while in the latter it would monitor client workstations. The system could also be placed within an organization and configured to monitor only accesses to certain web applications, which could be either local in the organization or hosted elsewhere.

The system is able to handle also encrypted HTTPS traffic. In case the target web application is not in the same administrative domain of our system, monitoring of HTTPS traffic requires the user to accept a self-signed certificate sent by the proxy in place of the certificate sent by the target application. For example, within our University, HTTPS connections to the exam registration system—an attractive target for credential stealing [19]—would not require any specific actions from the user, while connections to GMail or Facebook, as example, would involve accepting a self-signed certificate generated by the proxy—which, of course, needs to be part of the trusted computing base.

III. Mouse Dynamics

Each mouse-related DOM *event* e captured by Collector-C is composed of the following information: (i) timestamp, denoted $t(e)$; (ii) cartesian coordinates of the mouse position with respect to the browser viewport, denoted $x(e)$ and $y(e)$; (iii) event type $s(e)$, which can be one among click, double-click and movement. Collector-C captures events with a time resolution which depends on several factors, including the browser, the client processing power, and so on: we found in our experimentation that the mean time resolution was 25 ms and the median time resolution was 8 ms. Collector-C sends the collected events to Collector-S in batches, every few seconds. Collector-S extracts certain features from the collected events (Section III-A) and compares the observed mouse dynamics to the mouse dynamics of the claimed user (Section III-B).

A. Features extraction

Given the stream of events $(e_1, e_2, \dots)$ generated by each user, we construct a sequence $\mathcal{T}$ of trajectories. Each *trajectory* T represents how the user moved the mouse in a time span delimited by pauses and is built from the stream of events as follows: (i) we identify all the events e_k such that $t(e_{k+1}) - t(e_k) > 500\,\text{ms}$; (ii) we split the stream in one or more non-overlapping subsequences, each terminated by one of such events; (iii) we discard all subsequences composed of less than $N_T = 10$ events; (iv) in each remaining subsequence, we put all the N_T events immediately preceding the last event (included) in T and discard the other events; (v) we add T to $\mathcal{T}$ and sort $\mathcal{T}$ by the timestamp of the first event in each trajectory. In other words, all trajectories are composed of N_T events and we generate a new trajectory whenever there is a pause in the stream of mouse-generated events lasting at least 500 ms. We chose to split the mouse-generated stream of events based on pauses and to focus on the final part of each trajectory, based on the assumption that this segmentation allows keeping communication needs to a minimum while at the same time capturing the specific mouse-related tasks to be accomplished (i.e., menu selection, point-and-click, and so on).

We associate a feature vector $\mathbf{f}(T) \in \mathbb{R}^{39}$ with each trajectory $T = (e_1, \dots, e_{N_T})$, as follows. For each sample e_k in T, we compute:

- *direction* $d_k \in \{0, 45, 90, 135, \dots, 360\}$, computed as the direction of the vector $(x(e_k) - x(e_{k-1}), y(e_k) - y(e_{k-1}))$, rounded to multiples of 45 degrees; we set $d_k = 0$ if and only if the vector has exactly zero magnitude (i.e., the mouse did not move between e_{k-1} and e_k);
- *speed* $v_k = \frac{\sqrt{(x(e_k)-x(e_{k-1}))^2+(y(e_k)-y(e_{k-1}))^2}}{t(e_k)-t(e_{k-1})}$;
- *acceleration* $a_k = \frac{v_k - v_{k-1}}{t(e_k)-t(e_{k-1})}$.

We compute direction and speed excluding $k = 1$ and acceleration excluding $k = 1$ and $k = 2$.

The feature vector $\mathbf{f}(T)$ consists of the following features:

- duration of the trajectory, i.e., $t(e_{N_T}) - t(e_1)$;
- x-extent, i.e., $\max_{k=1}^{N_T} x(e_k) - \min_{k=1}^{N_T} x(e_k)$;
- y-extent, i.e., $\max_{k=1}^{N_T} y(e_k) - \min_{k=1}^{N_T} y(e_k)$;
- number of direction changes, i.e., the number of events e_k for which $d_k \neq d_{k-1}$;
- total covered distance, i.e., $\sum_{k=2}^{N_T} \sqrt{(x(e_k) - x(e_{k-1}))^2 + (y(e_k) - y(e_{k-1}))^2}$;
- average speed, i.e., $\frac{1}{N_T} \sum_{k=2}^{N_T} v_k$
- prevalent direction, i.e., the direction occurring most often;
- prevalent direction, without considering zero values;
- maximum distance between the event position (i.e., $x(e_k), y(e_k)$) and the straight line connecting $(x(e_1), y(e_1))$ to $(x(e_{N_T}), y(e_{N_T}))$;
- global direction, i.e., the direction of the vector $(x(e_1) - x(e_{N_T}), y(e_1) - y(e_{N_T}))$, rounded to multiples of 45 degrees as we did for d_k;
- distance between the first and the last position, i.e., $\sqrt{(x(e_1) - x(e_{N_T}))^2 + (y(e_1) - y(e_{N_T}))^2}$;
- average speed of the last five events, i.e., $\frac{1}{5} \sum_{k=N_T-4}^{N_T} v_k$;
- average speed of the first five events, i.e., $\frac{1}{5} \sum_{k=1}^{5} v_k$;
- counts of the 9 possible directions, i.e., for each n-th direction sector, with $n = 1, \dots, 9$ being the index for the set $\{0, 45, 90, 135, \dots, 360\}$, the n-th feature counts the number of events in T for which $d_k = n$;
- $N_T - 2$ features corresponding to the acceleration values $a_3, \dots, a_{N_T}$;
- $N_T - 1$ features corresponding to the speed values $v_2, \dots, v_{N_T}$.

We hence obtain a sequence F of feature vectors from $\mathcal{T}$,

which is itself obtained from the stream of events.

B. Detection methodology

We construct, in an initial off-line training phase, off-line a classifier for each authorized user to be used in the actual on-line continuous reauthentication phase.

In the *training phase*, we proceed as follows. Let U^- be the authorized user and $U_1^+, U_2^+, \ldots$ a set of other users. We collect a stream of events for each user and build the corresponding sequences of feature vectors F^-_{train} and $F^+_{1,\text{train}}, F^+_{2,\text{train}}, \ldots$, as explained in Section III-A. Next, we train a Support Vector Machine (SVM) binary classifier for the authorized user, the training data being composed of F^-_{train}, which contains training *negative* instances, and $F^+_{\text{train}} = F^+_{1,\text{train}} \cup F^+_{2,\text{train}} \cup \ldots$, which contains training *positive* instances. We truncate each $F^+_{i,\text{train}}$ so as to obtain a balanced training data, i.e., such that $|F^+_{\text{train}}| = |F^-_{\text{train}}|$ and $|F^+_{i,\text{train}}| = |F^+_{i,\text{train}}|, \forall i, j$. We denote the size of the training data by $n_{\text{train}} = |F^-_{\text{train}}| + |F^+_{\text{train}}|$.

We then compute a predefined threshold $\hat{w}$ based on the training data, as the mid-point between the False Rejection Rate (FRR) and the True Rejection Rate (TRR) on the training data, rescaled from $[0, 1]$ to $[0, w]$, w being a system parameter described below.

In the *reauthentication phase*, we proceed as follows. Let U^- be the claimed user (i.e., the one who successfully performed an initial authentication) and U the actual connected user who should be reauthenticated: the system collects the stream of events generated by U and constructs the corresponding sequence of feature vectors F. Whenever the system generates a new feature vector $\mathbf{f}$, i.e., whenever there is a pause larger than 500 ms in the mouse-generated stream of events, the system: (i) classifies $\mathbf{f}$ with the classifier trained for the claimed user U^-—we say that the classification outcome $\text{SVM}(\mathbf{f})$ is a *positive* if $\mathbf{f}$ does not fit the mouse dynamics profile for U^-; (ii) counts the number of positives generated in the last w classifications; and (iii) in case the number of positives exceeds the predefined threshold $\hat{w}$ (with $0 \leq \hat{w} \leq w$), generates an alert. Our filtering strategy has some similarity with the "trust" value associated with each individual classification in [12]. The cited work, though, provides an experimental assessment where a threshold value is chosen based on *all* the available data—including testing data—and does not detail the procedure which should be applied in practice, when operating solely on training data.

The technical details for informing Collector-S of which is the claimed user U^-, as well as those for selecting the corresponding classifier, are irrelevant to this discussion.

C. Discussion

Usage of mouse dynamics for continuous reauthentication involves solving a key design question. Let $M(U^-)$ denote the mouse dynamics profile for a certain user U^-. Shall the construction of $M(U^-)$ be based solely on data generated by U^-? In this respect, two approaches have been proposed: 1) labelled data of mouse dynamics generated by U^- and by all the other users $U_1^+, U_2^+, \ldots$ known to the system are used [7], [8]; or, 2) only data generated by U^- may be used [10], [11]. The rationale of approach 2 is that, in a system involving a large and dynamic set of users, assuming that the profile of each user requires data from every other user is not realistic. In this work, we followed approach 1, but we believe that the features available to our system would make approach 2 feasible as well.

In this respect, we note that, actually, a third option exists which deserves to be investigated, namely constructing $M(U^-)$ based on labelled data from U^- and from *some* users different from U^-, as opposed to *all* other users. We believe this option is worth exploring because, in our opinion, neither approach 1 nor approach 2 is intrinsically superior to the other. The essential issue is the separability of the features generated by U^- from those generated by impostors. In this respect, since an accurate representation of the mouse dynamics of impostors is not available, we believe it is not possible to tell a priori which of the two approaches results in better separability. In this work, we chose not to follow this third option because the available number of users is too small to perform a meaningful analysis—train the classifier for U^- with the users most similar to U^-, or those who are most different from U^-, or a randomly selected set.

As pointed out in the introduction, the threat model assumes an attacker who impersonates a legitimate user on a mouse-equipped platform in web browsing sessions which last for several minutes. In principle the model could address attackers who perform a short web session lasting just a few seconds, but we believe that in these cases the number of events available to the detection machinery would be too small to generate meaningful alerts, i.e., alerts associated with a reasonable level of accuracy. The threat model could be extended to partly address injection of fraudulent requests in web sessions generated by the legitimate user, for example through malware executing in the browser or in the client machine, or through hijacking of HTTP sessions from a different node. To this end, the system should learn the set of HTTP requests generated during mouse trajectories or shortly after them, and then it should generate an alert when those requests occur at time instants which are too far away from the time intervals of the observed mouse trajectories. Indeed, the ability to discriminate bot-generated traffic from human-generated traffic based solely on webpage-embedded loggers of keyboard and mouse events with excellent accuracy and negligible overhead, has been proven [20]. We have not addressed this extension in this work, though. Of course, an attacker capable of accurately mimicking the traffic generated by Collector-C in a web session originated by the victim user would be able to circumvent the system.

IV. Experimental evaluation

A. Dataset

We collected two datasets in different environments and operating conditions, one consisting of a stream of

events generated by 6 users and the other by 18 users. In both cases data were collected during normal working activity for several working days. Users of Dataset 1 were monitored for 4 weeks on the average and operated on different hardware equipment, in terms of screen size, screen resolution, actual mouse device—the data for each user being collected entirely on the same hardware. The average number of trajectories per user in Dataset 1 is $|\mathcal{T}| = 2927$. Users of Dataset 2 were monitored for 2 weeks and operated on homogeneous hardware equipment. The average number of trajectories per user in Dataset 2 is $|\mathcal{T}| = 3229$.

Concerning the way in which we collected the datasets, Dataset 1 reflects the scenario in which the system is deployed by a web application provider: in this scenario, legitimate users access the application mainly from their platform, whereas impostors will likely access from different platforms. Dataset 2 reflects the scenario in which the system is deployed within an organization where platforms are homogeneous.

It is important to remark that the user interacts with the browser and with other applications from which our machinery cannot collect mouse events. The considered scenario is thus more challenging than one in which all mouse events can be captured: this affects the trade-off between the classification accuracy which can be achieved and the observation time needed to achieve it. In our dataset, if we consider a *session* of interaction with the browser a time span without any pause longer than 10 min, then the average interval between consecutive trajectories within a session is $\bar{t} = 16.2$ s.

B. Procedure and results

For each user U_i in a dataset, (i) we built the sequence of events F_i^- and the set of sequences of events $F_j^+, i \neq j$, from the other users U_j in the dataset; (ii) we built the training data $F_{\text{train}}^-, F_{\text{train}}^+$ as described in Section III-B, trained the SVM classifier and computed $\hat{w}$; (iii) we applied the classifier to the sequence of feature vectors $F_{\text{test}}^- = F^- \setminus F_{\text{train}}^-$ and measured the False Rejection Rate (FRR); and (iv) finally, for each $j \neq i$, we applied the classifier to the sequence of feature vectors $F_{j,\text{test}}^+ = F_j^- \setminus F_{j\text{train}}^-$ and measured the False Acceptance Rate (FAR)—we thus simulated that U_j is an impostor. We repeated the above procedure 2 times for each user U_i by changing the training set composition, in order to average disadvantageous or advantageous random choices. We experimented on Dataset 1 only, Dataset 2 only, and union of Dataset 1 and Dataset 2—the latter only for one combination of w, n_{train}. That is, we tested the profile of each user against data of other 5, 17 and 23 users, respectively. We measured the performance in terms of FRR, FAR (averaged across the simulated impostors U_j for the sake of brevity) and accuracy (i.e., $1 - \frac{\text{FRR}+\text{FAR}}{2}$).

We experimented with $n_{\text{train}} \in \{1000, 1500, 2000\}$ and with $w = \{50, 100, 200, 350, 500\}$. The number of users was not uniform across all the experiments because, in some configurations, the amount of data available for some users was smaller than the amount of training data required by that configuration.

Table I
RESULTS WITH DIFFERENT VALUES OF n_{train} AND $w = 500$ FOR THE TWO DATASETS. TTT IS THE TIME TO TRAIN (SEE TEXT).

	TtT	Dataset 1			Dataset 2		
n_{train}	(h)	Acc.	FAR	FRR	Acc.	FAR	FRR
1000	4.5	91.4	8.9	8.2	88.2	9.7	14.0
1500	6.8	95.0	6.9	3.1	89.6	8.1	12.7
2000	9.0	96.5	6.1	0.8	92.2	9.5	6.1

Table II
RESULTS WITH DIFFERENT VALUES OF w AND $n_{\text{train}} = 2000$ FOR THE TWO DATASETS. TTD IS THE TIME TO DETECTION (SEE TEXT).

	TtD	Dataset 1			Dataset 2		
w	(min)	Acc.	FAR	FRR	Acc.	FAR	FRR
50	13.5	83.3	16.6	16.7	76.4	21.8	25.4
100	27.1	88.5	12.8	10.2	81.4	17.5	19.6
200	54.1	93.5	9.2	3.8	86.6	13.5	13.3
350	94.7	95.6	7.9	1.0	90.6	10.8	8.0
500	135.3	96.5	6.1	0.8	92.2	9.5	6.1

Table I shows the results for varying size of the training dataset n_{train}, with $w = 500$. As one would expect, increasing n_{train} indeed delivers better results. In a realistic scenario the user could be asked to train the system for a longer time to improve performances. The second column of Table I shows the Time to Train (TtT), which is the estimated time needed to collect the training data: TtT is computed as $n_{\text{train}}\bar{t}$ and represents the total duration of the sessions needed to collect n_{train} trajectories, one being generated each $\bar{t}$ s. The values (up to 9 h) for this figure appear to fit the considered application scenario.

Table II shows the results for varying size of the filtering window w, with $n_{\text{train}} = 2000$. It can be seen that increasing w improves accuracy significantly: in other words, the larger w, the smaller the variance of the system output. Of course, increasing w results in a longer time to detection (TdT, see second column in Table II, computed as $w\bar{t}$), because more trajectories (and hence more mouse-generated events) have to be observed by the system in order to output the first classification outcome on the connected user. For example, for $w = 350$ the accuracy is larger than 90% for both datasets and the time to detection is in the order of 1.5 h: this figure appears to be practical with respect to the considered threat model. It should be noted, however, that whenever at least w trajectories have been observed, a classification outcome is output soon after each trajectory.

It can be seen that the performance are better for Dataset 1. We think that this is due to the fact that users of Dataset 1 operated on different hardware platforms (see Section IV-A) resulting in observed trajectories (and corresponding feature vectors) which are intrinsically more separable. This finding fits the original aim of our proposal, since (i) Dataset 1 corresponds to the scenario of the system deployed by the web application provider and (ii) the architecture of our system is designed to accommodate this scenario, posing no requirements on the clients.

V. Concluding remarks

Usage of mouse dynamics as an authentication signal for complementing traditional authentication procedures and for constructing a further layer for a defense-in-depth strategy has been proposed several times in the past. We believe there are at least two major obstacles for a wide adoption of such approaches in practice. First, all published experimental evaluations considered a user base composed of only a few tens of users, hence the effectiveness of the approach over thousands or millions of users is still to be demonstrated. Second, earlier proposals assumed that client machines are instrumented for collecting all user-generated events of interest, which in many cases, including continuous reauthentication of accesses to web applications hosted in the cloud, is unfeasible.

Our work indicates that the latter issue may actually be overcome, thereby greatly improving the potential scope of mouse dynamics as a continuous reauthentication tool. We have shown a system for capturing GUI-related events for web traffic which does not require any specific software to be installed on client machines and is fully transparent to both users and web sites. We have also shown that, despite the intrinsic limitations of the collection procedure, the system exhibits accuracy in terms of FAR and FRR which is aligned with the state-of-the-art. The event capture machinery is sufficiently lightweight to not harm the actual user experience and may be deployed in a variety of scenarios, either close to the client or close to the server, and, perhaps most importantly, also in scenarios where clients and servers belong to distinct administrative domains. The details of the events actually captured could be easily modified to include, e.g., keystroke dynamics, and the detection procedure could be modified in order to take advantage of such additional information [5], [6]. We hope that the architectural advantages illustrated by our work may help in promoting further research aimed at understanding what can be realistically achieved on a very large user population.

Acknowledgments

The authors are grateful to Giorgio Davanzo and Daniele de Gan for their help in the data collection and earlier experimentation.

References

[1] D. Goodin, "360 million recently compromised passwords for sale online." http://arstechnica.com/security/2014/02/360-million-recently-compromised-passwords-for-sale-online/, Feb. 2014.

[2] A. Hern, "Did your adobe password leak? now you and 150m others can check," Nov. 2013.

[3] "Yahoo! japan says 22 MEELLION user IDs may have been nabbed." http://www.theregister.co.uk/2013/05/20/yahoo_japan_user_id_breach/, May 2013.

[4] R. Graham, "Confirmed: LinkedIn 6mil password dump is real." http://blog.erratasec.com/2012/06/confirmed-linkedin-6mil-password-dump.html#.Uxgxyfl5M0k, June 2012.

[5] D. Gunetti and C. Picardi, "Keystroke analysis of free text," *ACM Trans. Inf. Syst. Secur.*, vol. 8, pp. 312–347, Aug. 2005.

[6] I. Traore, I. Woungang, M. S. Obaidat, Y. Nakkabi, and I. Lai, "Online risk-based authentication using behavioral biometrics," *Multimedia Tools and Applications*, pp. 1–31, June 2013.

[7] A. Ahmed and I. Traore, "A new biometric technology based on mouse dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, pp. 165–179, July 2007.

[8] Y. Nakkabi, I. Traore, and A. Ahmed, "Improving mouse dynamics biometric performance using variance reduction via extractors with separate features," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, pp. 1345–1353, Nov. 2010.

[9] Z. Jorgensen and T. Yu, "On mouse dynamics as a behavioral biometric for authentication," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, 2011.

[10] C. Shen, Z. Cai, and X. Guan, "Continuous authentication for mouse dynamics: A pattern-growth approach," in *2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–12, 2012.

[11] C. Shen, Z. Cai, X. Guan, Y. Du, and R. Maxion, "User authentication through mouse dynamics," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 16–30, 2013.

[12] S. Mondal and P. Bours, "Continuous authentication using mouse dynamics," in *Biometrics Special Interest Group (BIOSIG), 2013 International Conference of the*, pp. 1–12, Sept. 2013.

[13] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann, "Touch Me Once and I Know It's You!: Implicit Authentication Based on Touch Screen Patterns," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 987–996, 2012.

[14] L. O'Gorman, "Comparing passwords, tokens, and biometrics for user authentication," *Proceedings of the IEEE*, vol. 91, pp. 2021–2040, Dec. 2003.

[15] J. Bonneau, "Authentication is machine learning," *Light Blue Touchpaper - Security Research, University of Cambridge*, Dec. 2012.

[16] "Same origin policy." http://www.w3.org/Security/wiki/Same_Origin_Policy.

[17] "JSMeter: characterizing real-world behavior of JavaScript programs," Tech. Rep. MSR-TR-2009-173, Microsoft Research, 2009.

[18] "ISPs scramble to explain mouse-sniffing tool." http://www.theregister.co.uk/2013/08/27/isps_scramble_to_explain_away_mousesniffing/, Aug. 2013.

[19] J. Hawes, "Jail time for university hacker who changed his grades to straight as." http://nakedsecurity.sophos.com/2014/02/28/jail-time-for-university-hacker-who-changed-his-grades-to-straight-as/, Feb. 2014.

[20] Z. Chu, S. Gianvecchio, A. Koehl, H. Wang, and S. Jajodia, "Blog or block: Detecting blog bots through behavioral biometrics," *Computer Networks*, vol. 57, Feb. 2013.