Bot or Human? A Behavior-Based Online Bot Detection System

Zi Chu[1], Steven Gianvecchio[2], and Haining Wang[3](✉)

[1] Airbnb, San Francisco, CA 94117, USA
[2] MITRE, Hampton, VA 23666, USA
[3] University of Delaware, Newark, DE 19716, USA
hnw@udel.edu

Abstract. The abuse of Internet online services by automated programs, known as bots, poses a serious threat to Internet users. Bots target popular Internet online services, such as web blogs and online social networks, to distribute spam and malware. In this work, we will first characterize the human behaviors and bot behaviors in online services. Based on the behavior characterization, we propose an effective detection system to accurately distinguish bots from humans. Our proposed detection system consists of two main components: (1) a client-side logger and (2) a server-side classifier. The client-side logger records user behavioral events such as mouse movement and keystroke data, and provides this data in batches to a server-side classifier which identifies a user as human or bot. Our experimental results demonstrate that our proposed detection is able to achieve very high accuracy with negligible overhead.

1 Introduction

Interactive Internet applications like online blogs have become popular in the past decade. These applications enable interactive communications among Internet users, in terms of text messages. Millions of people around the world use these interactive applications to exchange information and discuss a broad range of topics on-line. Such applications are different from conventional networked applications, because of their human-to-human interaction and low bandwidth consumption. However, the large user base and open nature of the Internet make them ideal targets for malicious exploitation.

Currently the most common form of malicious exploit and the most difficult to thwart, is the use of automated programs known as bots to automatically perform human tasks on online applications. Bots have been found on a number of online systems, including online blogging and online social networking. Bots exploit these on-line systems to send spam, spread malware, and mount phishing attacks. The abuse of online services by bots has caused serious damages and posed serious threats to on-line users.

So far, the efforts to combat bots have focused on two different approaches: (1) content-based filtering and (2) human interactive proofs (HIPs). The content-based filters, used by third party clients, suffer from high false negative rates because bot makers frequently update bots to evade the filtering rules. The use of human interactive proofs, such as CAPTCHAs [11, 15], is also ineffective because bot operators can assist bots in

P. Samarati et al. (Eds.): Jajodia Festschrift, LNCS 11170, pp. 432–449, 2018.
https://doi.org/10.1007/978-3-030-04834-1_21

passing the tests to log in [22,23]. Thus, multiple CAPTCHA tests are needed throughout a session to block the login of bots; otherwise, an adversary can pass the one-time test and log a bot into the session. However, although multiple tests can foil the adversary's attempt for bot login, they are too obtrusive and distractive for a regular human user to tolerate as well.

In this work, we introduce a new approach based on human observational behaviors (HOBs) for detecting and blocking bots. In particular, our proposed approach exploits behavioral biometrics, including mouse and keystroke dynamics, for bot detection. HOBs offer two distinct advantages over conventional detection methods like HIP-based approach. First, HOPs provide continuous monitoring throughout an online session. Second, HOPs are non-interactive, i.e., no test is presented to a user, making HOPs completely non-obtrusive. HOPs differentiate bots from human users by passively observing those tasks that are difficult for bots to perform in a human-like manner. Therefore, adversaries must resolve a difficult problem in human behavior modeling to evade a well-designed HOP. However, modeling human behavior is known to be very difficult, as shown in behavioral biometric research [10,12,26]. Moreover, the HOP approach is not based on any single metric of the human behavior, but rather a collection of different kinds of behavioral metrics.

Based on the proposed HOB approach, we build a prototype of an automatic classification system that detects bots. The system consists of two components, a client-side logger and a server-side detector. The client-side logger is implemented as a JavaScript snippet that runs in the client browser. It records a user's input actions during her stay at the site and streams the data to the server-side detector. The detector processes raw user input (UI) data, and extracts biometrics-related features. The core of the detector is a machine-learning-based classifier which is tuned with training data for the binary classification, namely determining whether the user is human or bot. To validate the efficacy of our proposed HOB approach, we conduct a case study in the detection of blog bots for online blogging systems. We collect user input activities on a real, active blog site. By measuring and characterizing biometric features of user input data, we discover the fundamental differences between human and blog bot in how they surf web pages and post comments. Our experimental results demonstrate an overall detection accuracy greater than 99% with minor overhead.

2 Behavior Characterization

In this section, we analyze user behaviors, namely how a user surfs blog pages and posts comments, based on data collected from a large corpus of users. We first introduce three types of blog bots, then describe how we collect user input data from a blog site. Finally, we characterize the behavioral differences between human and blog bot, in terms of keystroke and mouse dynamics.

2.1 Blog Bots

Fundamentally, current blog bots can be categorized into three different types based on their working mechanisms: Form Injection Bot, Human Mimic Bot, and Replay Bot.

Form Injection Bots do not post comments via the browser. Rather, it directly sends an HTTP request to the server for the blog page where it plans to post comments. After receiving the HTML content of the requested page, it analyzes the HTML structure of the comment form. Then, it injects content into form fields[1], constructs a syntactically legal HTTP response with the HTML form data as the body, and sends it to the submission URL at the server. To evade the server's check on the HTTP response, the bot often forges certain fields in the response header, such as Referer, User Agent, and Cookie. Furthermore, some bots are equipped with CAPTCHA deciphering capability to crack the CAPTCHA defense. However, they do not generate any mouse or keystroke events. Currently this type of bot is the most widely used blog bot in cyberspace [5].

Contemporary detection methods have realized the importance of detecting human activities during the form filling procedure. A server only accepts a user as human if mouse or keyboard events are detected. Thus, bot authors are motivated to create a more advanced bot type, namely the Human Mimic Bot. These bots open a blog page in the browser, and use OS API calls to generate keystroke and mouse events. In this manner, it mimics human browsing behavior, fooling older detection methods. For example, the bot strolls down the page to the bottom by repetitively sending "Press down-key" commands. Then, it moves the mouse cursor into each field of the comment form, and types in prepared text content by sending a sequence of keystrokes. Finally, the bot posts the comment by generating a mouse click on the submit button. The server cannot distinguish whether the UI events are generated via hardware (such as the mouse device and keyboard) or via software (such as Human Mimic Bot) by merely checking the received user input data. The server will be deceived by Human Mimic Bot if it only relies on the presence of UI events for bot detection.

Some research into behavioral biometrics has found out that human behavior is more complex than bot behavior. Compared with the inherent irregularity and burstiness of human behavior, bots exhibit regular patterns of limited variety [17]. For example, many bots move the mouse cursor in straight lines at a constant speed, or strike keys with even intervals. Such perfect regular actions cannot be achieved by human. Thus, the server could detect Human Mimic Bot by taking behavioral complexity into account. With high fidelity of mimicry, Replay Bots are more advanced than Human Mimic Bots, and are probably the most difficult to detect among contemporary blog bots. When a human is filling a form, Replay Bot records her actions. Later on, it impersonates the human by replaying recorded traces on form submission pages. The standard interfaces utilized by popular blogs and message boards, such as WordPress or vBulletin, make such replay attacks possible.

To characterize the bot behaviors, we use existing bot tools or libraries to configure the three types of blog bots. The Form Inject Bot is implemented as a PHP cURL script. The comment form at our blog site is submitted via the POST method. The cURL script assigns every input field with an appropriate value, encapsulates the form data into a string, and submits it to the PHP script at the server that processes the form. We configure the Human Mimic Bot based on the AutoHotkey script [2], which is an open-

[1] The form is usually well-structured, and the ID/name of each input field remains constant. For example, <input type="text" name="email" /> is the text field to enter email address. Thus, the bot author programs the bot to recognize fields and fill in appropriate content.

source Windows program designed for automating the Windows GUI and for general scripting[2]. We customize the script for our blog site, and thus it can generate actions corresponding to the page layout[3]. The script mimics all kinds of normal human actions, such as moving and clicking the mouse cursor, scrolling the page up and down, drag-and-dropping an area, and typing keys. To simulate various effects, we assign action parameters with different constants or random values. Taking mouse movement as an example, we change endpoint coordinates and movement speed to generate different traces. For keystrokes, we change the duration (the length of time the key is held) and inter-arrival time (the time from pressing one key to another) to generate different typing rhythms. We choose the Global Mouse and Keyboard Library for Windows [6] as the Replay Bot in our experiments, which has both record and replay capabilities. The record and replay are implemented using the mouse and keyboard APIs in Windows. Specifically, for recording, global hooks are created to capture keyboard and mouse events; and for replaying, the keybd_event and mouse_event APIs in Windows are used.

2.2 UI Data Collection

For client-side monitoring, we develop a logger written in JavaScript, which is embedded in the header template of every webpage, and in this way it records UI data during the user's entire visit at the site. The user behavior is in constant monitoring, which prevents bots from bypassing routing checkpoints (such as CAPTCHA recognition during login). More specifically, five raw UI events generated by the user in the browser are collected, including Key Press, Key Release, Mouse Move, Mouse Button Press, and Mouse Button Release. The logger streams the UI data to the server for further processing and classification. More details of the logger implementation are presented in Sect. 3.1. Note that no user sensitive data content (e.g., password) is recorded by our logger. We have also obtained the approval from the Institutional Review Board (IRB) of our university, which ensures the appropriate and ethical use of human input data in our work.

The collection of human UI data is described as follows. We collected data from a busy blog site consisting of over 65,000 members. The site averages 800 simultaneous online users, and in order to prevent spam, the site requires visitors to register with real credentials and log in before posting content. Content is manually reviewed by site administrators, moderators, and a community of dedicated users. Should an account post spam and be reported, the associated content is quickly removed and the account gets suspended. We collected data from 1,078 distinct signed-in site members during several two-hour monitoring sessions on a single day. The data collection was completely transparent to users, and the interactions consist of both reading and posting of content. Our real-world data with the large user population covers a wide range of human input behavior. The data also presents an advantage over the lab environment

[2] There are other similar bot tools that may generate simple human behavior, such as AutoIt [3] and AutoMe [4].

[3] The page layout is different from page to page, and may affect how the Human Mimic Bot works. For example, by moving down the same amount of pixels, the mouse enters the comment form on one page, but falls out of the form on another page.

tests, where a user's performance might be at odds with her normal behavior. We maintain a high degree of confidence that the users in this dataset are indeed human, as their registrations are manually screened by site administrators, and posted content is screened by a community of users, resulting in a low overall observed incidence of spam.

Table 1. User input actions

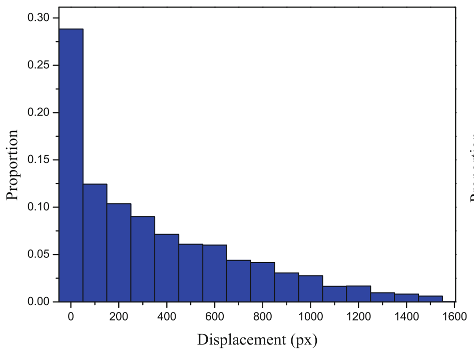
Action	Description
Keystroke	The press and release of the same key
Point	A set of continuous mouse moves with no mouse clicks, and the interval between two consecutive moves is no more than 0.4 s
Click	The press and release of the same mouse button
Point-and-Click	A point followed by a click within 0.4 s
Drag-and-Drop	Mouse button down, movement, and then mouse button up

Correspondingly, we run three types of blog bots to collect bot input data. By including username and password to the POST data body, Form Inject Bots can post comments. As it does not open a webpage in the browser to generate any input events, the server does not receive any UI data. Thus, Form Inject Bots can be easily detected. We also run multiple instances of the Human Mimic Bot, and each instance is assigned with different settings (such as varied typing rhythms and mouse movement speeds) to generate different behavior. We generate the traces of Human Mimic Bot for 30 h. We run the Replay Bot for six rounds, which last for 2 h in total. In each round, a human user fills in the comment form, and Replay Bot records the human trace and replays it.

Lastly, we explain the reasons that we run customized bots in the controlled "sand box" to generate bot input data. First, ground truth creation and data collection is an example of the chicken or the egg causality dilemma. We must know the true identity of a user to label it as human or bot in the ground truth set. In other words, we cannot collect data in the wild and recognize what data are generated by bot or not. After being trained on the ground truth set, the classifier can distinguish between human and bot. Second, we do not create bots. Instead, we customize bots based on existing tools and libraries without changing their mechanisms. The authenticity of bot input data is reserved. In addition, a bot needs to be customized to operate on a specific blog site[4], and no existing tools can be generative to all blogs.

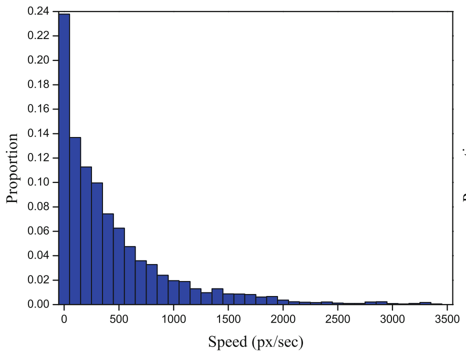
Raw UI events cannot efficiently describe user browsing activities. We develop a parser to integrate raw events into compound actions as shown in Table 1. For example,

[4] For example, the position of the submit button may vary in the webpage layout. The bot must be customized to move to the button and generate a click event on it.



(a) Human (b) Human Mimic Bot

Fig. 1. Displacement for Point-and-Click



(a) Human (b) Human Mimic Bot

Fig. 2. Speed for Point-and-Click

the Key Press event and the following Key Release event of the same key is integrated as a Keystroke action, and a set of continuous Mouse Move events are grouped as a Point action.

2.3 UI Data Measurements

Based on the collected UI data from human and bot, we analyze the keystroke and mouse dynamics and characterize different behavioral patterns for humans and bots, respectively. For the profiling of bot behavior, we only use the traces of Human Mimic Bot, and exclude those of Form Inject Bot and Replay Bot[5].

Figures 1 and 2 illustrate two mouse kinematics features, displacement and speed, for the Point-and-Click action, respectively. In Fig. 1 with the bin resolution of 100 pixels, we observe that human users generate far more displacements with short length

[5] Form Inject Bot generates no UI data. As Replay Bot replays traces generated by human, it is inappropriate to include human traces to characterize bot behavior.

than with long length. About 60.64% of displacements are less than 400 pixels, while only 8.52% are greater than 1000 pixels. In contrast, bots tend to move the mouse at all displacements. Figure 2 with the bin resolution of 100 pixels per second shows the movement speed of bot is faster than that of human. The average speed of bot is 1520.83 pixels per second in our observation, but the average speed of human is 427.43 pixels per second. Furthermore, human speed is limited within 3500 pixels per second, due to the physical movement constraints of human wrist and arm. Finally, we observe that some bots move the mouse at fixed speeds.

Figure 3 shows the mouse movement efficiency for the Point-and-Click action, with the bin resolution of 0.02 s. For a mouse movement from the starting point to the end point, displacement is the segment length between the two points, and distance is the actual length traversed. Movement efficiency is defined as the ratio of displacement over distance. Straight line movement has the highest efficiency at 1. The more curvy the movement is, the lower its efficiency is. Our first observation is that bots move the mouse cursor with much greater efficiency than humans. About 59.23% of bot movements achieve efficiency greater than 0.94, while only 28.60% of human movements are equally efficient. As the Point action is the integration of a set of continuous raw Mouse Move events, we could have treated several segments of Move event as the curve of Point action, which lowers the bot efficiency during the calculation. Thus, there could have been more bot movements with the efficiency of 1 (namely, straight movement). Our second observation is that, the probability of human movement efficiency follows a lognormal (3P) distribution in our dataset[6], and the bot probability does not fit any well-known distributions. For humans, most movements are curves, since it is physically difficult to generate perfect straight lines over certain length or time.

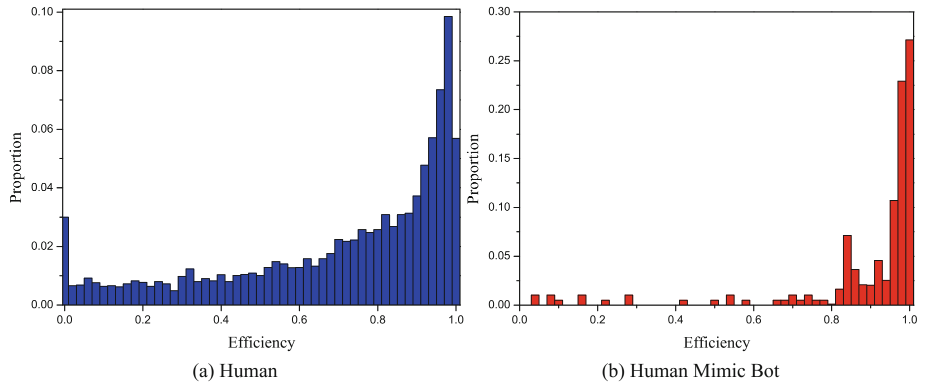


Fig. 3. Movement efficiency for Point-and-Click

Figure 4 shows the distribution of inter-arrival times for the Keystroke action, with a bin resolution of 0.05 s. We make two observations from the figure. First, bots strike

[6] Kolmogorov-Smirnov test presents P-value of the distribution fitting at 0.882 with a 99% confidence level.

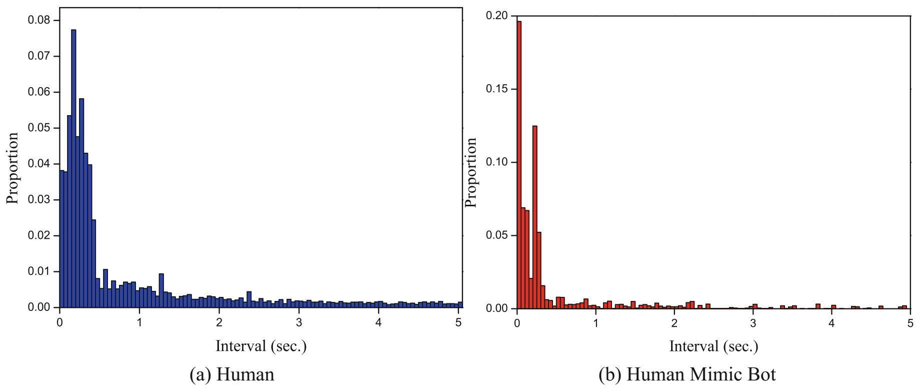


Fig. 4. Inter-arrival time distribution for Keystroke

keys obviously faster than humans. About 21.49% of bot keystrokes are less than 0.05 s, and only 5.82% of human keystrokes are issued within that range. A human user has to look up keys on the keyboard, and moves her fingers to hit keys. Physical movements cannot compete with keystroke events generated by software. Second, for bots, the probabilities of intervals at 0.05 and 0.25 s are greatly higher than other values. This implies that some bots may use periodic timers to issue keystrokes at fixed intervals.

We also observe similar distribution patterns of Keystroke duration between human and bot. The keystroke duration is the elapsed time between a key press and its corresponding release. The distribution patterns are similar with those in Fig. 4. Bots hold keys much shorter than humans. While 45.42% of bot keystrokes are held less than 0.3 s, only 23.11% of human keystrokes are within that range. A human needs time to move his finger up to release the key after he presses it down. In addition, for bots, the probability of intervals between 0.05 and 0.15 s are greatly higher than other values. The periodic timer may set fixed intervals between consecutive key press and release events. Due to the space limit, the related figures are not included in the chapter.

3 System Design

Our detection system is mainly composed of the webpage-embedded logger and the server-side detector. The logger collects UI activities in the client browser and sends data to the server. The detector analyzes the UI data of a user and decides whether it is human or bot. The high-level system architecture is shown in Fig. 5.

3.1 Webpage-Embedded Logger

As mentioned in Sect. 2.2, the logger is implemented as JavaScript code, and embedded in every webpage of the blog site. As a result, JavaScript is required by the blog site and non-JavaScript clients are blocked from posting or must pass a conventional HIP, such as a CAPTCHA. When a user visits the blog, the logger runs silently inside the client browser. It is totally transparent to the user, and no extensions need to be installed. The logger collects five raw UI events generated by the user inside the browser, including Key Press, Key Release, Mouse Move, Mouse Button Press, and Mouse Button Release. Each event is associated with a JavaScript listener. After an event happens, the listener is triggered to generate a record in the JSON format [7]. Every record has several fields to describe the event attributes[7]. The polling rate of the logger is decided by the client operating system, and is generally high enough to capture UI events. For example, in Windows 7, the polling rate is 125 Hz, namely polling every 8 ms. The logger buffers the collected events within a small time window, and then sends the data in a batch to the server via Ajax (Asynchronous JavaScript and XML). The asynchronous communication mechanism helps save network traffic between server and client, as no additional traffic occurs when no events happen within the window. Besides, according to Sect. 4.2, only a certain number of user actions are needed to correctly classify a user. It also helps reduce network traffic.

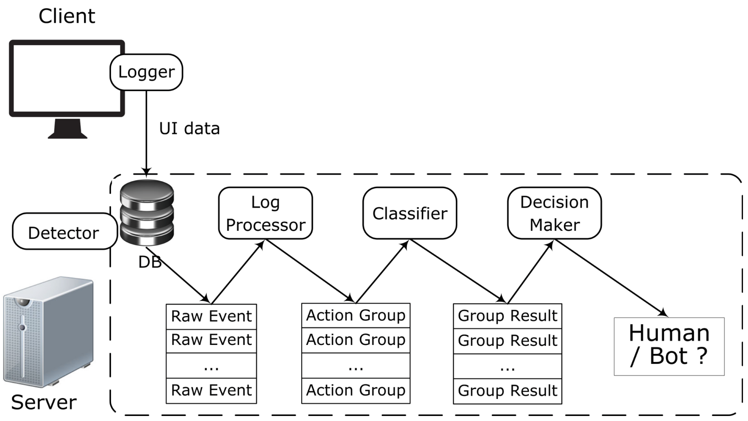


Fig. 5. Detection system architecture

As our detection method is generic to other types of form bots, such as those automatically perform massive account registration and online voting, we need to address the privacy and security concerns of using the logger to collect user input data. First, we discuss the user privacy protection. As the logger is implemented as JavaScript code running in web pages of the blog site, it is strictly constrained by the same-origin policy

[7] Take the following Mouse Move record as an example, {“time”:1278555037098, “type”:“Mouse Move”, “X”:590, “Y”:10, “tagName”:“DIV”, “tagID”:“footnote”}. The “time” field contains the time stamp of the event in the unit of millisecond. The two coordinates, X and Y, denote the mouse cursor position. The last two fields describe the name and ID of the DOM element where the event happens, such as <div ID=“footnote”>. In a record of Mouse Press, {“time”:1278555074750, “type”:“Mouse Press”, “virtualKey”:0x01, “tagName”:“HTML”}, The “virtualKey” field denotes the virtual-key code of 0x01 in hexadecimal value, which corresponds to the left mouse button here.

[19] enforced by the browser, and thus cannot access content of other sites or programs. This makes it very different from the OS-level keyloggers. In other words, our logger can only access the data that a user generates on the blog, which will be submitted to the blog site anyway. Thus, the logger does not endanger user privacy. Second, we consider the confidentiality of user input content transferred over the Internet. When a user types in content on the webpage, the key values of strokes are recorded in the format of virtual-key codes [9]. The link between the logger and the server is not encrypted. To prevent an eavesdropper from intercepting data packages in plain text and recovering the user input content, the logger replaces each key value of strokes with a wildcard character. This wildcard replacement enforces the confidentiality of user input content, and avoids the additional overhead by encryption.

3.2 Server-Side Detector

The detector consists of three components: the log processor, the classifier, and the decision maker. The UI data of each user is processed by the log processor, which converts raw events into high-level actions and encapsulates an adjustable number of consecutive actions to form action groups. The classifier processes each action group in the user log and assigns it with a classification score, indicating how likely the action group is generated by human or bot. Finally, the decision maker determines the class of the user based on the classification results of action groups. Each of the components is explained as follows.

Log Processor. When the UI data arrives at the server, it is in the format of raw events, such as Mouse Move and Key Press. The raw data is stored at the back-end MySQL database, and can be easily grouped per user who generates the data. Before classifying a user, the log processor processes the user log by converting raw events into high-level UI actions defined in Table 1. Furthermore, the log processor calculates the timing entropy of intervals of the whole raw event sequence in the user log, which detects periodic or regular timing of the entire user behavior.

The human behavior is often more complicated than that of bot [13,16], which can be measured by entropy rate. It is a measure of the complexity of a process [14]. A high entropy rate indicates a random process, whereas a low indicates a regular process. The entropy rate is defined as the conditional entropy of an infinite sequence. As our real dataset is finite, the conditional entropy of finite sequences is used to estimate the entropy rate. For estimation, we use the corrected conditional entropy [24], which is defined as follows.

A random process $X = \{X_i\}$ is defined as a sequence of random variables. The entropy of such a sequence is defined as:

$$E(X_1, ..., X_n) = -\sum_{i=1}^{n} P(x_1, ..., x_n) \log P(x_1, ..., x_n), \tag{1}$$

where $P(x_1, ..., x_n)$ is the joint probability $P(X_1 = x_1, ..., X_n = x_n)$.

Thus, the conditional entropy of a random variable is:

$$E(X_n \mid X_1, ..., X_{n-1}) = E(X_1, ..., X_n) - E(X_1, ..., X_{n-1}). \tag{2}$$

Then the entropy rate of a random process is defined as:

$$\overline{E}(X) = \lim_{n \to \infty} E(X_n \mid X_1, ..., X_{n-1}). \tag{3}$$

The corrected conditional entropy is computed as a modification of Eq. 3. First, the joint probabilities, $P(X_1 = x_1, ..., X_n = x_n)$ are replaced with empirically-derived probabilities. The data is binned into Q bins, i.e., values are converted to bin numbers from 1 to Q. The probabilities are then determined by the proportions of bin number sequences in the data. The entropy estimate and conditional entropy estimate, based on empirically-derived probabilities, are denoted as EN and CE, respectively. Second, a corrective term, $perc(X_n) \cdot EN(X_1)$, is added to adjust for the limited number of sequences for increasing values of n [24]. The corrected conditional entropy, denoted as CCE, is computed as:

$$\begin{aligned} &CCE(X_n \mid X_1, ..., X_{n-1}) = \\ &\quad CE(X_n \mid X_1, ..., X_{n-1}) + perc(X_n) \cdot EN(X_1) \end{aligned} \tag{4}$$

Based on Eq. 4, we calculate the CCE of intervals of the raw event sequence for a user as the timing entropy.

Finally, a set of classification features are generated for every action, which are listed in Table 2. They are used by the machine-learning based classifier for bot detection. More specifically, we group raw UI events into an action record as shown in Table 1. For example, a "Point" action contains a set of mouse move events. The value of duration feature is the timestamp difference between the last and first mouse move events. Similarly, the value of distance feature is the actual length traversed by all the mouse move events. The former seven features are directly retrieved from the action itself. In particular, the first four features are the basic ones, while average speed and move efficiency are derived from them[8]. These two derived features reveal the inherent correlation among features and accelerate the tree building. The last feature is the timing entropy of the whole event interval sequence of a user, not of a single action. An action only consists of several events, which are too few to extract timing regularity. It is statistically meaningful to calculate entropy at the user level. We include the entropy feature in the action record to inform the classifier the behavioral timing pattern of the user who generates the action.

Classifier. Our classifier is based on the C4.5 algorithm [20] that builds a decision tree for classification. The decision tree predicts the class of an unknown sample based on the observed attributes. There are two types of nodes in the decision tree, the leaf node labeled with the class value (such as human or bot), and the interior node that corresponds to an attribute and links to a subtree. The tree is constructed by dividing the

[8] Average speed is distance over duration, and move efficiency is displacement over distance.

Table 2. Classification features of user actions

Feature	Description
Duration	Mouse/keystroke actions
Distance	Mouse actions
Displacement	Mouse actions
Displacement angle	Mouse actions
Average speed	Mouse actions
Move efficiency	Mouse actions
Virtual key value	Left/middle/right button for mouse actions, and a wildcard character for keystrokes
Timing entropy	Event interval sequence of the target user

training dataset into subsets based on the attribute value test. This partitioning process is executed on each derived subset in a recursive manner. The fundamental ideas behind C4.5 are briefly described as follows. The tree is built from the root downward to leaves. During the construction path, each interior node must be associated with the attribute that is most informative among the attributes not yet included in the path. C4.5 uses entropy to measure how informative an attribute is. Given a probability distribution $P = \{p_1, p_2, ..., p_n\}$, the entropy of *P* is defined as

$$E(P) = -\sum_{i=1}^{n} p_i \log p_i, \tag{5}$$

We denote *D* as the dataset of labeled samples, and *C* as the class with *k* values, $C = \{C_1, C_2, ..., C_k\}$. The information required to identify the class of a sample in *D* is denoted as *Info(D) = E(P)*, where *P*, as the probability distribution of *C*, is

$$P = \{\frac{|C_1|}{|D|}, \frac{|C_2|}{|D|}, \ldots, \frac{|C_k|}{|D|}\}. \tag{6}$$

If we partition *D* based on the value of an attribute *A* into subsets $\{D_1, D_2, \ldots, D_m\}$,

$$Info(A, D) = \sum_{i=1}^{m} \frac{|D_i|}{|D|} Info(D_i). \tag{7}$$

After the value of attribute *A* is obtained, the corresponding gain in information due to *A* is denoted as

$$Gain(A, D) = Info(D) - Info(A, D), \tag{8}$$

As *Gain* favors attributes that have a large number of values, to compensate for this the C4.5 algorithm uses *Gain Ratio* as

$$GainRatio(A, D) = \frac{Gain(A, D)}{SplitInfo(A, D)} \tag{9}$$

where *SplitInfo(A,D)* is the information due to the splitting of D based on the value of attribute A. Thus,

$$SplitInfo(A, D) = E(\frac{|D_1|}{|D|}, \frac{|D_2|}{|D|}, ..., \frac{|D_n|}{|D|}) \tag{10}$$

The gain ratio is used to rank how informative attributes are and to construct the decision tree, where each node is associated with an attribute having the greatest gain ratio among the attributes not yet included in the path from the root. In other words, C4.5 applies a greedy search by selecting the candidate test that maximizes the heuristic splitting criterion.

We choose the C4.5 algorithm for the classification due to the following four reasons. First, it builds the decision tree in an efficient manner by processing a large amount of training data in a short time. Furthermore, the tree is robust even if assumptions, to some extent, are violated by the real data model. Second, it uses the white box model, which is easy to understand and interpret by boolean logic. Third, C4.5 is capable of processing both continuous and discrete values (such as numerical and categorical data), which is an improvement from the earlier ID3 algorithm [25]. Last, after the tree creation, C4.5 prunes the tree from top down with attempts to constrain the tree height and avoid overfitting.

We use J48 as implementation, which is an open source Java program of the C4.5 algorithm in the Weka data mining tool [18]. Each action record is in such a format of feature vector as <*duration, distance, displacement, displacement angle, average speed, move efficiency, virtual key value, timing entropy*>, listed in Table 2. The J48 classifier takes input from all actions in an action group[9], and outputs the classification result indicating whether the action group is generated by human or bot.

Decision Maker. The user log contains multiple action groups, and each group is determined by the classifier as generated by either human or bot. The decision maker presents the summary of the classifications of UI actions over a period of time by employing the majority voting rule. More specifically, if the majority[10] of action groups are classified as human, then the user is classified as human, and vice versa. Since classification on individual actions cannot always be accurate, the more actions are included, the more confident the final decision is.

4 Evaluation

In this section we evaluate the efficacy of our detection system in terms of detection accuracy, detection time, and induced system overhead.

[9] Input is converted the ARFF format required by Weka [1].

[10] As our classification only involves two categories, human and bot, a majority means more than half of the votes.

4.1 Experimental Setup

Our experiments are based on 239 h of user traces, including 207 h of human and 32 h of bot[11]. The traces are collected from more than 1,000 human users and two types of blog bots (namely Human Mimic Bot and Replay Bot). The details about user composition are described in Sect. 2.2. In summary, the user input dataset consists of 4,520,165 raw events, which are further converted into 190,677 compound actions.

Table 3. True positive and negative rates vs No. of actions per group

Actions per group	TPR	TNR
2	0.974	0.9993
4	0.9945	0.9996
6	0.9865	0.9989
8	0.9879	0.9989

We use *cross validation* with ten folds [21] to train and test the classifier on our UI dataset. The dataset is randomly partitioned into ten complementary subsets. In each round, one of the ten subsets is retained to validate the classifier (as the test set), while the remaining nine subsets are used to train the classifier (as the training set). Every round is an independent procedure, as the classifier is reset at the beginning of the round and then re-trained. The test results from ten rounds are averaged to generate the final estimation. The advantage of cross validation is that, all the samples in the dataset are used for both training and validation and each sample is validated exactly once.

4.2 System Performance

Our detection system has two adjustable parameters that affect the system performance: (1) the number of actions per group and (2) the total number of actions required to correctly classify a user. We describe the configuration procedure of each parameter as follows.

We set different values for the number of actions per group, run cross validation tests, and then calculate the true positive rate (TPR)[12] and true negative rate (TNR)[13] for each value. The results are listed in Table 3. During the classification, the classifier treats a group of actions as one entity[14], and produces the classification result for the group, not for individual actions. In our experiment, the setting of four generates the

[11] The idle time is not included in the traces. The bot trace consists of 30 h of Human Mimic Bot data and 2 h of Replay Bot data.

[12] The true positive rate is the ratio of the number of bots which are correctly classified to the number of all the bots.

[13] The true negative rate is the ratio of the number of humans which are correctly classified to the number of all the humans.

[14] A series of consecutive actions represent continuous behavior well.

highest TPR and TNR among all the values. Therefore, we set the number of actions per group as four.

The second parameter, the total number of actions required to correctly classify a user, directly affects the system performance in terms of detection accuracy and detection time. Generally speaking, the more actions observed from the user, the more accurate the classification result will be. On the other hand, processing more actions costs more time and increases the detection time. Given the number of actions per group is four, we run experiments with cross validation on the whole ground truth to determine how many actions are required to achieve a high accuracy. The results are summarized in the column labeled as "Both Bots" in Table 4. Since each action group is configured to contain four actions, the total number of actions required equals the group number multiplied by four. The last row in Table 4 labeled as "Entire" corresponds to the baseline case, in which the classifier takes all the actions in the user log as input. It is used as upper-limit for accuracy comparison. We can see that the detection accuracy in terms of TPR and TNR increases as the total number of actions processed by the classifier increases. With the group number as 24 (namely $24 * 4 = 96$ actions in total), TPR and TNR are very close to those of the entire log. Besides, the accuracy gain increases very slowly after the group number exceeds 24. Thus, the system is configured to process 24 action groups while each group includes 4 actions. Each group is labeled as either human or bot, and the user is eventually classified as the category with more labels using the majority voting rule. For example, if the action group sequence is labeled as <human, human, bot, human, $\cdots$, human>, then the user is classified as human. The C4.5 algorithm generates a decision tree based on our dataset and prunes it afterwards. The construction procedure costs 4.96 s, and returns a tree with 57 nodes. The tree consists of 29 leaves and 28 interior nodes including the root. The overall detection accuracy is 0.9972 with the root mean squared error at 0.0244.

Table 4. True positive and negative rates vs Number of groups

Group no	Both Bots		Human Mimic Bot		Replay Bot	
	TPR	TNR	TPR	TNR	TPR	TNR
4	0.6975	0.9972	0.7016	0.998	0.6359	0.9992
8	0.7673	0.9956	0.7710	0.9982	0.7117	0.9974
12	0.8172	0.9973	0.8198	0.9991	0.7781	0.9982
16	0.8788	0.9978	0.8802	0.9992	0.8578	0.9986
20	0.917	0.9982	0.9208	0.9994	0.8599	0.9988
24	0.9794	0.9983	0.9817	0.9996	0.9448	0.9987
Entire	0.9945	0.9996	0.9964	0.9999	0.9660	0.9997

The detection time is mainly decided by the total number of actions processed by the classifier. The average time per action is less than one millisecond. The overall time cost per user, including log processing and classification, is averagely 3.2 s.

We speculate whether one bot type is more difficult to detect than the other. Thus, we separate the evaluation on Human Mimic Bot and Replay Bot to see how accurately our system can detect the two types of blog bots. More specifically, we derive two subsets of the ground truth: one with the entire trace of human and Human Mimic Bot, and the other with that of human and Replay Bot. The results are displayed in the last two columns in Table 4. We have two observations. Firstly, for each row, the TPR of Human Mimic Bot is greater than that of Replay Bot. It is easier to detect Human Mimic Bot thanks to the simplicity and regularity of its behavior. Due to certain implementation deficiencies of the Replay Bot tools, our system also effectively detects Replay Bot with the TPR greater than 0.966. Secondly, the TNR is greater than the corresponding TPR for every bot type. In other words, the FNR is greater than the FPR. It reflects our design philosophy that, the system may miss capturing some bots, but it seldom mis-classifies human as bot to upset legitimate users.

4.3 System Overhead

As the detector is employed on the server side, it must be light-weight and scalable enough to accommodate numerous concurrent user classifications. We estimate the additional overhead induced by the detector for the case, in which 10,000 users access the server simultaneously.

In terms of network bandwidth consumption, the logger streams the user input data in the JSON format to the server. An average user generates a trace at a size around 200 Kbytes. Then, the aggregated network bandwidth consumed at the server-side for receiving UI data is about 4.2 Mbps. Considering the wide deployment of Gigabit Ethernet, this network bandwidth requirement can be easily met.

The main memory cost at the server side is to accommodate user input actions and the decision tree outputs for each user. An input action contains eight features, and each feature occupies 5 bytes, except the virtual key value with 2 bytes. Thus, a single action consumes 37 bytes. Each action group contains 4 actions, and is assigned with a result that occupies 1 byte. The detector only needs 24 action groups from the user log for classification, and thus classifying a single user consumes up to 3.49 Kbytes of memory. Scaled to 10,000 online users, the memory cost of the server will be 34.1 Mbytes, which is very affordable for a modern server.

The computational overhead is also very minor. We run J48 in the Weka, a Java implementation of the C4.5 algorithm, on a workstation with an Inter Core 2 Duo 2.4 GHz CPU. The classification time is 10.85 s for the traces of 239 h.

5 Conclusion

This chapter presents a bot detection system, which leverages the behavioral differences between human users and bots in their mouse and keystroke activities. Compared to conventional detection methods based on Human Interactive Proofs, such as CAPTCHA, our detection system does not require additional user participation, and is thus both transparent and unobtrusive to users. We have collected real user input traces of 239 h from a busy blog site. Based on these real UI traces, we have discovered different user behavioral characteristics, and further developed useful features for classification. Our detection system consists of a client-side logger and server-side detector. The logger passively collects user activities and streams this data to the server. The detector processes the log and identifies whether it is generated by human or bot. The core of our detection system is a statistical classifier (i.e., C4.5 algorithm) that builds a decision tree. It takes the action stream as input, and classifies the user by the majority voting rule. We perform a set of experiments to tune the system parameters and evaluate the system's performance. The experimental results show that the overall detection accuracy is higher than 99%. The additional overhead induced by the detection is minor in terms of CPU and memory costs.

References

1. Attribute-relation file format (arff). http://www.cs.waikato.ac.nz/ml/weka/arff.html
2. Autohotkey - free mouse and keyboard macro program with hotkeys. http://www.autohotkey.com/
3. Autoit, automation and scripting language. http://www.autoitscript.com/site/autoit/
4. Autome - automate mouse and keyboard actions. http://www.asoftech.com/autome/
5. Blogbot by incansoft. http://blogbot.auto-submitters.com/
6. Global mouse and keyboard library. http://www.codeproject.com/KB/system/globalmousekeyboardlib.aspx
7. Json, javascript object notation. http://www.json.org/
8. Ultimate wordpress comment submitter. http://www.wordpresscommentspammer.com/
9. Virtual-key codes. http://msdn.microsoft.com/en-us/library/ms927178.aspx
10. Ahmed, A.A.E., Traore, I.: A new biometric technology based on mouse dynamics. IEEE Trans. Dependable Secure Comput. **4**(3), 165–179 (2007)
11. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: using hard AI problems for security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 294–311. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_18
12. Van Balen, N., Ball, C.T., Wang, H.: A behavioral biometrics based approach to online gender classification. In: Deng, R., Weng, J., Ren, K., Yegneswaran, V. (eds.) SecureComm 2016. LNICST, vol. 198, pp. 475–495. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59608-2_27
13. Chu, Z., Gianvecchio, S., Wang, H., Jajodia, S.: Who is tweeting on Twitter: human, bot or cyborg? In: Proceedings of the 2010 Annual Computer Security Applications Conference, Austin, TX, USA (2010)
14. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley-Interscience, New York (2006)
15. Funk, C., Liu, Y.: Symmetry reCAPTCHA. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), Las Vegas, NV, USA, June 2016
16. Gianvecchio, S., Wang, H.: Detecting covert timing channels: an entropy-based approach. In: Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, VA, USA, October–November 2007
17. Gianvecchio, S., Wu., Z., Xie, M., Wang, H.: Battle of botcraft: fighting bots in online games with human observational proofs. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, Chicago, IL, USA (2009)
18. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. **11**, 10–18 (2009)

19. Jackson, C., Bortz, A., Boneh, D., Mitchell, J.C.: Protecting browser state from web privacy attacks. In: Proceedings of the 15th International Conference on World Wide Web, pp. 737–744 (2006)
20. Kohavi, R., Quinlan, R.: Decision tree discovery. In: Handbook of Data Mining and Knowledge Discovery, pp. 267–276. University Press (1999)
21. McLachlan, G., Do, K., Ambroise, C.: Analyzing Microarray Gene Expression Data. Wiley, Hoboken (2004)
22. Mohta, A.: Bots are back in Yahoo! chat rooms. http://www.technospot.net/blogs/bots-are-back-in-yahoo-chat-room/
23. Mohta, A.: Yahoo! chat adds CAPTCHA check to remove bots. http://www.technospot.net/blogs/yahoo-chat-captcha-check-to-remove-bots/
24. Porta, A., et al.: Measuring regularity by means of a corrected conditional entropy in sympathetic outflow. Biol. Cybern. **78**(1), 71–78 (1998)
25. Quinlan, J.R.: Discovering Rules from Large Collections of Examples: A Case Study. Edinburgh University Press, Edinburgh (1979)
26. Zheng, N., Bai, K., Huang, H., Wang, H.: You are how you touch: user verification on smartphones via tapping behaviors. In: Proceedings of IEEE Conference on Network Protocol (ICNP 2014), Research Triangle Park, NC, USA, October 2014