Computer Networks 57 (2013) 634–646

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Blog or block: Detecting blog bots through behavioral biometrics

Zi Chu [a,*], Steven Gianvecchio [a], Aaron Koehl [a], Haining Wang [a], Sushil Jajodia [b]

[a] Department of Computer Science, The College of William and Mary, Williamsburg, VA 23187, USA
[b] Center for Secure Information Systems, George Mason University, Fairfax, VA 22030, USA

ARTICLE INFO

Article history:
Received 19 May 2012
Received in revised form 18 August 2012
Accepted 10 October 2012
Available online 17 October 2012

Keywords:
Blog Bot
Behavioral biometrics
Automatic classification
Security
Web

ABSTRACT

Blog bots are automated scripts or programs that post comments to blog sites, often including spam or other malicious links. An effective defense against the automatic form filling and posting from blog bots is to detect and validate the human presence. Conventional detection methods usually require direct participation of human users, such as recognizing a CAPTCHA image, which can be burdensome for users. In this paper, we present a new detection approach by using behavioral biometrics, primarily mouse and keystroke dynamics, to distinguish between human and bot. Based on passive monitoring, the proposed approach does not require any direct user participation. We collect real user input data from a very active online community and blog site, and use this data to characterize behavioral differences between human and bot. The most useful features for classification provide the basis for a detection system consisting of two main components: a webpage-embedded logger and a server-side classifier. The webpage-embedded logger records mouse movement and keystroke data while a user is filling out a form, and provides this data in batches to a server-side detector, which classifies the poster as human or bot. Our experimental results demonstrate an overall detection accuracy greater than 99%, with negligible overhead.

1. Introduction

Blogs (from *weblog*), are a popular application of Web 2.0. Internet users publish articles on blog sites, such as personal online diaries or news on a particular subject. Like normal web pages, blog pages are primarily textual combined with images, videos, and links. The distinctive feature of blog is user interaction, which allows visitors to leave comments to blog articles. A visitor fills in the comment form and submits it, and his comment will display below the article in reverse-chronological order. Unfortunately, the increasing popularity of blogs and the simplicity of posting comments have made it easy for blog bots to automatically post comments with malicious intent. According to the estimation of [1], about 83 percent of blog comments are injected by blog bots, indicating how rampant blog bots are in the blogosphere. Most of these automated comments are associated with spam websites, containing either traceback links to inflate search engine rankings [2], or other content to lure visitors to these sites.

Since the majority of content generated by blog bots is unwanted by blog owners and visitors, blogging software has incorporated a variety of methods to discourage posting from these sources. Fundamentally, detecting human presence is an effective defense against blog bots. Conventional detection methods based on Human Interactive Proofs (HIPs) [3] usually require direct participation from human users, such as CAPTCHA. As a reverse Turing test, it challenges a user with an image carrying alphanumeric text. The user must enter the exact text before the blog site can accept the comment for submission. To cope with an advanced bot's capability for image recognition (namely, De-CAPTCHA), CAPTCHA tools add image noise to the background canvas, and greatly distort characters [4]. However, such a CAPTCHA validation also requires non-trivial effort

* Corresponding author. Tel.: +1 917 698 5015.
E-mail addresses: zichu@cs.wm.edu (Z. Chu), srgian@cs.wm.edu (S. Gianvecchio), amkoeh@cs.wm.edu (A. Koehl), hnw@cs.wm.edu (H. Wang), jajodia@gmu.edu (S. Jajodia).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.10.005

from human users. In some cases, users have to try several times to correctly recognize a CAPTCHA image because it has become more and more difficult even for human to read. Such a validation in place may effect a significant decrease in participation from human visitors.

In this paper, we present a new method based on passive monitoring for blog bot detection, as conventional detection systems have become a nuisance for human users. Our proposed approach employs behavioral biometrics, including mouse and keystroke dynamics, to distinguish between human and bot. It has two major advantages over existing solutions. First, it uses continuous monitoring throughout the entire user session, and eliminates single checkpoints. In contrast, blog sites deployed with the conventional detection face the dilemma of applying one-time test or multiple tests. On one hand, blog bots can pass the one-time test, such as account login, with the help of human. On the other hand, multiple tests, such as recognizing a CAPTCHA image before each comment posting, are too intrusive for human users. Our passive continuous monitoring resolves the above dilemma. Second, our method is non-interactive and completely transparent to users. Moreover, no detection decision needs to be made until the user submits the comment, which in turn saves system resources. We develop a passive, webpage-embedded logger to 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. These results greatly facilitate accurate detection of blog bots.

We build a prototype of an automatic classification system that detects blog bots based on user input data. The system consists of two components, a webpage-embedded logger and a server-side detector. The logger is implemented as a JavaScript snippet that runs in the webpage on 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. Informed with the classification result, the server decides whether or not to accept the comment form submission.[1] We evaluate the efficacy of the detection system by conducting a series of experiments over the user input dataset. The experimental results demonstrate that the system can detect 97.9% of current blog bots with extremely low false positive rate of 0.2%.

As defense against bots is a challenging task, we acknowledge that our detection alone cannot eliminate the problem. However, our approach is a significant complement to conventional HIPs. We believe that, with the inherent irregularity and complexity of human behavior, it is extremely difficult if not impossible for a bot to completely mimic human behavior. Our behavior-based detection raises the bar for bot participation during this game of cat-and-mouse.

The remainder of the paper is organized as follows. Section 2 covers related work on blog bot and behavioral biometrics. Section 3 details our measurements and characterization of user inputs from human visitors and blog bots, respectively. Section 4 describes our automatic classification system. Section 5 evaluates the system efficacy for detecting blog bots. Section 6 discusses potential evasion against our detection system. Finally, Section 7 concludes the paper.

2. Background and related work

From the perspective of blog content creation, there are two types of blog bots. The first type is the article posting bot, which automatically publishes blog articles. For example, it pipelines RSS feeds from other sites as articles into the blog site, or posts preset content for a spam blog (also known as a splog). Since the posting of articles usually requires the elevated privilege of the webmaster, article posting bots are not the focus of this study. The second type is the comment posting bot, which posts comments or replies to blog sites. Given a link to a blog site, this bot analyzes the HTML structure of the blog article, especially the "leave a comment" form, fills in input fields, and posts a comment automatically. Most blog sites do not require visitors to register to post comments, and thus give ample space for bots to exploit. The focus of our work is on this bot type, and the term "blog bot" in the remainder of the paper implicitly refers to comment posting bots. Currently, blog bots are mainly created to fulfill two tasks. First, the bot posts a comment with a backlink directing to a specific website (such as that of the bot owner).[2] Posting backlinks to numerous blog sites has the effect of increasing the search engine traffic, in an attempt to boost search rankings for the originating site. The search industry has already employed some mitigation measures, such as Google's no-follow tag to prevent spam from polluting on search rankings. However, bots still massively generate inflation backlinks due to the ease and low cost of posting. Second, bots post comments with spam content (also known as spam comments) aiming to lure visitors to spam-related or other malicious sites.[3] Many blog sites eliminate spam comments based on content filtering, and Akismet [5] is such a distributed anti-spam web service. Each time a new comment is posted to the blog, it is submitted to Akismet, which checks content, runs other tests, and returns the spam detection result to the blog. Our work has the different research direction, and checks posting behavior instead of content posted.

2.1. Existing web bot detection

There have been many previous works on web bot detection. Stassopoulou et al. [6] introduced a probabilistic

[1] For instance, the server can be configured to accept the manual submission from human, and reject the automated form completion from bot.

[2] Here is an example of backlink comment, "*I don't really think this is right, but believe whatever you want. The real story can be found here on my blog: http://myblog.com/blog/*".

[3] Here is an example of spam comment, "*Thousands of cheap replica watches and fashionable designer bands at www.hot-replica.com/*".

modeling approach for web bot detection by analyzing server access logs. They constructed a Bayesian network that classifies log sessions as being crawler or human induced. Their classifier uses some features to characterize crawler and human behaviors, including maximum sustained click rate, session duration, percentage of image requests, pdf/ps requests, 4xx error responses, and robots.txt file requests. Tan et al. [7] did a similar study by investigating navigational patterns of web bots. They extracted features from server logs such as total number of pages requested, average time between two HTML requests, and percentage of requests made with GET/POST methods, which are useful for a machine learning algorithm to distinguish between human and bot. Park et al. [8] took web bot detection as a special form of the Turing test and defended the system by inferring whether the traffic source is human or bot. More specifically, their detection decision depends on evidence of mouse movement or keyboard activity from the client. If no human input activities are detected by a web server, the user is classified as bot.

However, all these existing detection mechanisms are only effective for detecting form-injection bots that do not generate any human-like activities. They miss more advanced bots such as human-mimic bots, which can navigate web pages in the browser by generating simple user input actions. A web server cannot easily detect this type of bot by using navigational patterns or logs. Moreover, since human-mimic bots send mouse and keystroke actions to the browser, the detector in [8] is deceived by the existence of forged human activity. Different from the previous detection research, our proposed approach does not merely depend on the presence of mouse or keystroke actions to distinguish human from bot. Our solution extracts features from UI actions that represent the inherent irregularity of human behaviors, and applies these features for bot detection.

2.2. Behavioral biometrics

The fundamental idea of our approach is to exploit behavioral biometrics for bot detection. Biometrics-based authentication is defined as the automated use of a collection of factors describing human behavioral or physiological characteristics to establish or verify a precise identity [9]. It can be classified into two categories: physiological biometrics and behavioral biometrics. Physiological biometrics uses measurements from the human body, including fingerprints, iris, retina, and facial scanning, and so on. Behavioral biometrics uses measurements based on human actions, such as signatures, voice and keystroke dynamics. Compared with physiological biometrics, normally behavioral biometrics do not require any special-purposed hardware for data collection, and are easy to employ.

Among all behavioral biometrics, mouse and keystroke dynamics are the most common metrics attempted for on-line user authentication [10–12]. Keystroke dynamics measures duration (the length of time a key is pressed down) and inter-arrival time (the time from pressing one key to another) for keystroke actions. In the previous works [10,13,11], the way that a user types at the keyboard is analyzed to identify its habitual typing rhythm and patterns. Mouse dynamics measures the characteristics of mouse actions of an individual user when it is interacting with the graphical user interface (GUI). Raw events generated by the mouse input device include cursor movement, mouse button press and release. In [12], high-level mouse actions are defined as the four meaningful combinations of raw events: Mouse-Move (i.e., general mouse movement), Drag-and-Drop (i.e., the action starts with mouse button down, movement, and then button released), Point-and-Click (i.e., mouse movement followed by a click or a double click), and Silence (i.e., no movement). Using neural networks, Ahmed et al. [12] modeled the mouse dynamics characteristics from the captured user input data. They implemented a detector that generates a signature for a user. User identification is conducted by comparing two signatures.

Our paper extends behavioral biometrics into blog bot detection, mainly using keystroke and mouse dynamics. In the context of blog user behavior characterization, keystroke and mouse dynamics are complementary to each other. This is because human users move the mouse cursor to surf blog pages, and strike the keyboard to post comments. However, our work significantly differs from aforementioned biometric detectors. Our work distinguishes two classes of users, human and blog bot, instead of identifying individual users. Some previous work indicates that, behavioral biometrics may generate non-negligible errors in identifying individuals as one's behavior may vary significantly [14]. Our evaluation results demonstrate behavioral biometrics works accurately for the problem of classifying two classes: bot and human, instead of user identification.

The closest previous work to ours is [15], which also applies behavioral biometrics for detecting game bots in online games. On one hand, blog bot detection is different from game bot detection, due to different application environments.[4] In [15], the user input actions are collected by the game client, while our work resorts to JavaScript in the blog page for user input collection. Neural networks are used in [15], while our work uses decision tree for classification. Decision tree is more efficient than neural networks on our dataset of user input activities, and the tree structure clearly presents how features are weighted during the classification. On the other hand, behavioral biometrics works well for both cases, and user input behaviors remain consistent in different online applications, either online blogging or online gaming.

3. 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.

[4] Game bot operates on a map and fulfills a series of game-related missions.

3.1. 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,[5] 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 [16,17].

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 [15]. 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 [18], which is an open-source Windows program designed for automating the Windows GUI and for general scripting.[6] We customize the script for our blog site, and thus it can generate actions corresponding to the page layout.[7] 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 [21] 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.

3.2. 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 Section 4.1. Note that no user sensitive data content (e.g., password) is recorded by our logger. Besides, user data is anonymized by hashing user ids. In other words, we do not track the UI data back to its generator. 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.

[5] 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.

[6] There are other similar bot tools that may generate simple human behavior, such as AutoIt [19] and AutoMe [20].

[7] 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.

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 1078 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 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.

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,[8] 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, 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.

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

3.3. 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.[9]

Figs. 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 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. Fig. 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.

Fig. 3 shows the mouse movement efficiency for the Point-and-Click action, with the bin resolution of 0.02 second. 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

[8] 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.

[9] 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.

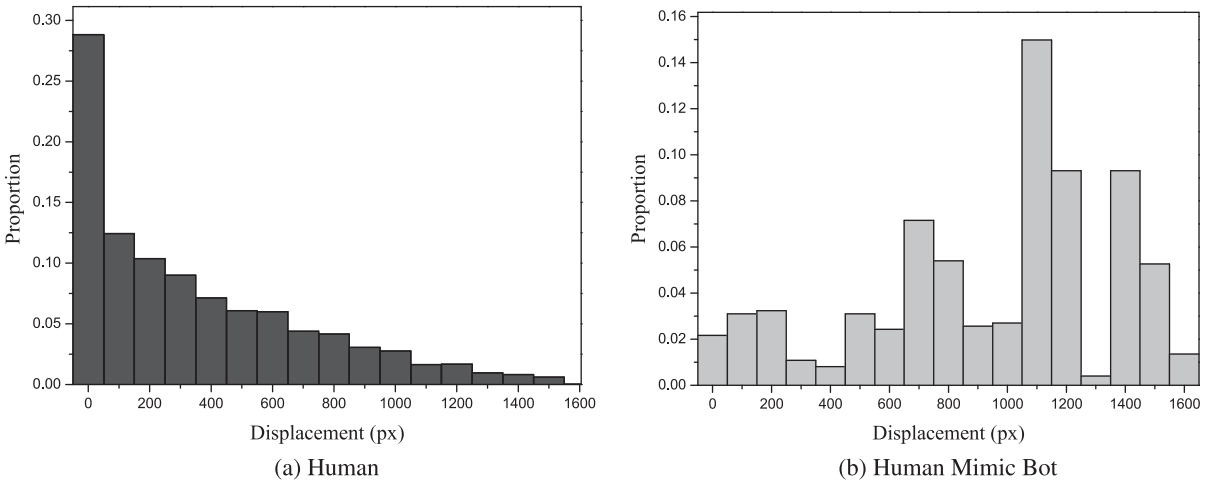

Fig. 1. Displacement for Point-and-Click.

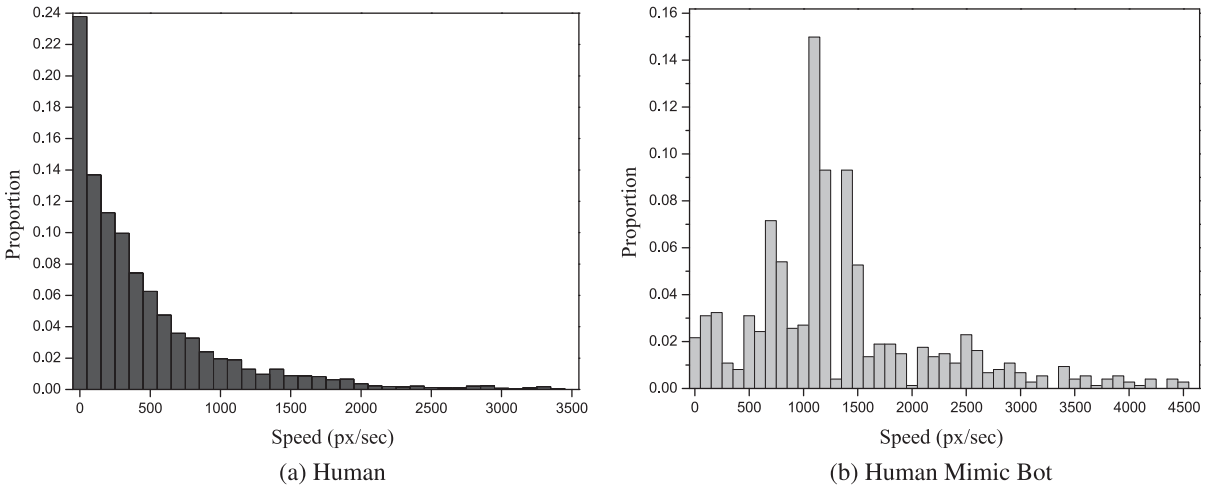

Fig. 2. Speed for Point-and-Click.

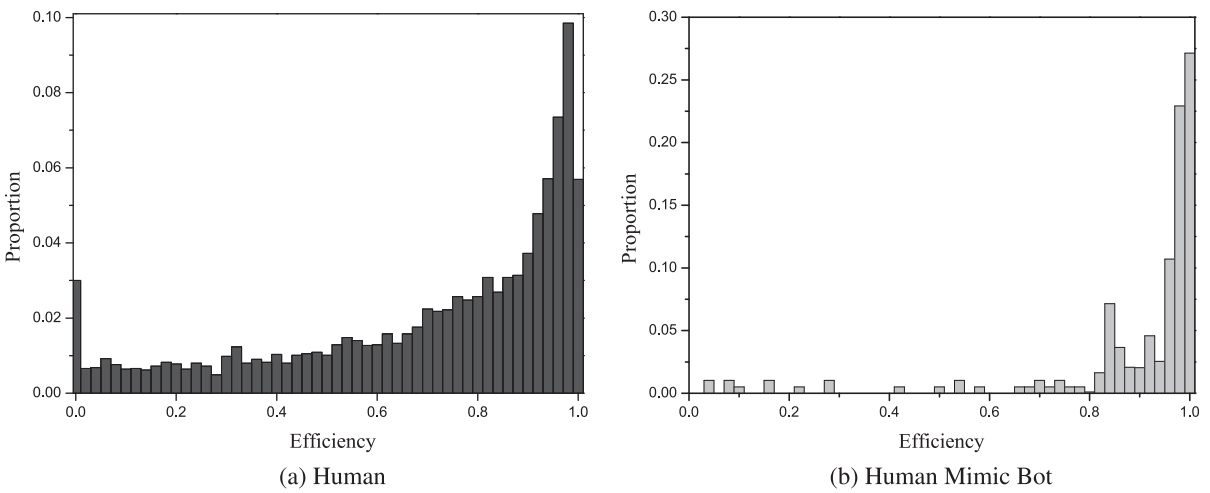

Fig. 3. Movement efficiency for Point-and-Click.

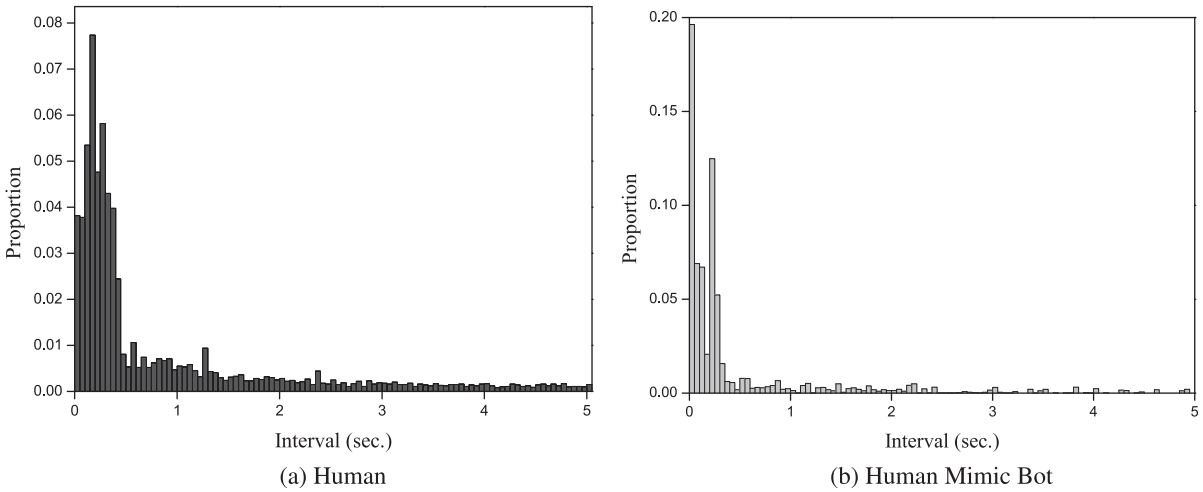

Fig. 4. Inter-arrival time distribution for keystroke.

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,[10] 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. 4 shows the distribution of inter-arrival times for the Keystroke action, with a bin resolution of 0.05 second. We make two observations from the figure. First, bots strike keys obviously faster than humans. About 21.49% of bot keystrokes are less than 0.05 second, 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 seconds 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 second, 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 seconds 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 paper.

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

4. 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.

4.1. Webpage-embedded logger

As mentioned in Section 3.2, 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 [22]. Every record has several fields to describe the event attributes.[11] 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 125Hz, namely polling every 8 milliseconds. The logger buffers the collected events within a small time window,

[11] 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 . 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.

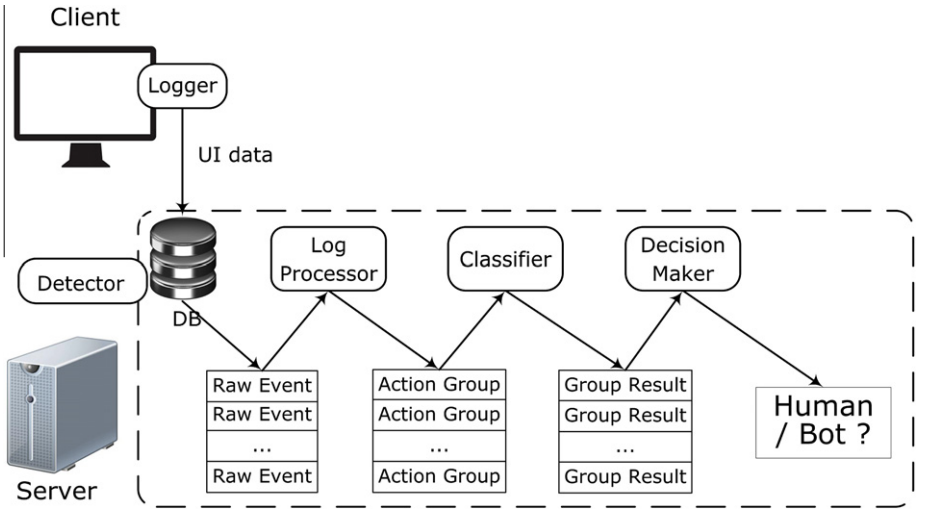

Fig. 5. Detection system architecture.

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 Section 5.2, only a certain number of user actions are needed to correctly classify a user. It also helps reduce network traffic.

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 [23] 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 [24]. 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.

4.2. 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.

4.2.1. 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 [25,26], which can be measured by entropy rate. It is a measure of the complexity of a process [27]. 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 [28], 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:

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

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

Thus, the conditional entropy of a random variable is:

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

$$E(X_n|X_1,\ldots,X_{n-1}) = E(X_1,\ldots,X_n) - E(X_1,\ldots,X_{n-1}). \quad (2)$$

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

$$\bar{E}(X) = \lim_{n\to\infty} E(X_n|X_1,\ldots,X_{n-1}). \quad (3)$$

The corrected conditional entropy is computed as a modification of Eq. (3). First, the joint probabilities, $P(X_1 = x_1,\ldots,X_n = x_n)$ 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[28]. The corrected conditional entropy, denoted as *CCE*, is computed as:

$$CCE(X_n|X_1,\ldots,X_{n-1}) = CE(X_n|X_1,\ldots,X_{n-1}) + perc(X_n)\cdot EN(X_1) \quad (4)$$

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.[12] 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.

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

4.2.2. Classifier

Our classifier is based on the C4.5 algorithm [29] 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 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 $P = \{p_1, p_2, \ldots, p_n\}$, the entropy of P is defined as

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

We denote D as the dataset of labeled samples, and C as the class with k values, $C = \{C_1, C_2, \ldots, C_k\}$. 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

$$P = \left\{\frac{|C_1|}{|D|}, \frac{|C_2|}{|D|}, \ldots, \frac{|C_k|}{|D|}\right\}. \quad (6)$$

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

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

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

$$Gain(A, D) = Info(D) - Info(A, D), \quad (8)$$

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

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

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

$$SplitInfo(A, D) = E\left(\frac{|D_1|}{|D|}, \frac{|D_2|}{|D|}, \ldots, \frac{|D_n|}{|D|}\right) \quad (10)$$

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 [30]. 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 [31]. 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,[13] and outputs the classification result indicating whether the action group is generated by human or bot.

4.2.3. 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[14] 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.

5. Evaluation

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

5.1. Experimental setup

Our experiments are based on 239 h of user traces, including 207 h of human and 32 h of bot.[15] The traces are collected from more than 1000 human users and two types of blog bots (namely Human Mimic Bot and Replay Bot). The details about user composition are described in Section 3.2. In summary, the user input dataset consists of 4,520,165 raw events, which are further converted into 190,677 compound actions.

We use *cross validation* with ten folds [33] to train and test the classifier on our UI dataset. The dataset is randomly partitioned into 10 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.

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

5.2. 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)[16] and true negative rate (TNR)[17] for each value. The results are listed in Table 3. During the classification, the classifier treats a group of actions as one entity,[18] and produces the classification result for the group, not for individual actions. In our experiment, the setting of four generates the 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 in-

[13] Input is converted the ARFF format required by Weka [32].
[14] As our classification only involves two categories, human and bot, a majority means more than half of the votes.
[15] 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.
[16] The true positive rate is the ratio of the number of bots which are correctly classified to the number of all the bots.
[17] The true negative rate is the ratio of the number of humans which are correctly classified to the number of all the humans.
[18] A series of consecutive actions represent continuous behavior well.

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

creases. 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 seconds, 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.

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,[19] 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 misclassifies human as bot to upset legitimate users.

5.3. 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 four 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.

6. Discussion

Once attackers know the existence of our detection system, they will attempt to evade it. An adversary could directly send synthetic traces to the server, trying to deceive the detector. We discuss the trace forging in three aspects. First, the adversary records the trace when a human fills in the form, and replays it. Some current record-and-replay tools, such as the Global Mouse and Keyboard Library for Windows [21], cannot restore human trace with high fidelity and thus create detectable artifacts. For example, the timing of events is more regular in the replayed trace than the original human trace, which can be detected based on its regular and low entropy pattern. The difference can be identified by our detector.

Second, suppose the bot tool would perfectly replay human trace. Then, the adversary has to record a different

[19] More explanations are made in Section 6.

trace in real-time (typing them out by a person) for each different spam message it wants to post. Therefore, our detection system will at least significantly raise the bar against blog bots and their spamming cost. Third, as an alternative of replay, the adversary might be motivated to develop a complete generative model of human user-input dynamics, and send the bogus trace to the server bypassing the client-side logger. However, the inherent complexity and uncertainty of human behavior makes it a very difficult modeling problem, because no such a model exists yet according to our best knowledge.

Now we discuss the limitation that enabling JavaScript in a browser is required for the blog sites protected by our system. With many popular sites, such as Twitter, Facebook, and Digger, being entirely dependent on JavaScript for key tasks [34] such as posting, we do not feel this requirement will cause any problem for most human users. According to a study of Internet user browser settings [35], only a tiny proportion (around 0.56%) of human users turn off JavaScript in practice. In other words, most human users will successfully generate UI data to the logger and pass our behavior detection even without noticing it. Bots could disable JavaScript to prevent the logger from obtaining their genuine traces. However, the disabled JavaScript will reveal the high likelihood of a bot being behind the browser. Meanwhile, to cope with this situation, a server can take one step back and resume the use of the conventional HIP methods, such as CAPTCHA, to defend against bots.

Lastly, we discuss the feasibility of applying our work to other interactive web applications. The core idea of this paper is detecting human participation by analyzing the user input data, and it can be applied to those web applications that require human participation, such as forums and online social networks. However, developers need to fulfill some tasks during transplantation. They need to collect the ground truth data for their own applications, as user behavior (both of human and bot) may be application specific. Some features may also have to be replaced for accurate classification.

7. Conclusion

This paper presents a blog 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 webpage-embedded 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 the C4.5 algorithm that builds a decision tree. It takes the action stream as input, and classifies a user by the majority voting rule.

We have performed 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 over 99%. The additional overhead induced by the detection is minor in terms of CPU and memory costs. Since the detection only requires user input traces, our method can be applied to detecting other types of form bots, such has account registration and online voting.

Acknowledgments

We are grateful to the anonymous referees for their insightful feedback. This work was partially supported by ARO grant W911NF-11-1-0149 and NSF grant 0901537.

References

[1] Sophos Security Threat Report, 2010 <http://www.sophos.com/sophos/docs/eng/papers/sophos-security-threat-report-jan-2010-wpna.pdf> (accessed 08.03.2012).
[2] J.-H. Kim, T.-B. Yoon, K.-S. Kim, J.-H. Lee, Trackback-rank: an effective ranking algorithm for the blog search, in: Proceedings of the Second International Symposium on Intelligent Information Technology Application, vol. 03, Washington, DC, USA, 2008, pp. 503–507.
[3] K. Chellapilla, K. Larson, P. Simard, M. Czerwinski, Designing human friendly human interaction proofs (hips), in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2005.
[4] J. Yan, A.S. El Ahmad, A low-cost attack on a microsoft captcha, in: Proceedings of the 15th ACM Conference on Computer and Communications, Security, 2008, pp. 543–554.
[5] Akismet, Comment Spam Prevention for Your Blog <http://akismet.com/> (accessed 08.03.2012).
[6] A. Stassopoulou, M.D. Dikaiakos, Web robot detection: a probabilistic reasoning approach, Comput. Netw. 53 (2009) 265–278.
[7] P.-N. Tan, V. Kumar, Discovery of web robot sessions based on their navigational patterns, Data Min. Knowl. Discov. 6 (2002) 9–35.
[8] K. Park, V.S. Pai, K.-W. Lee, S. Calo, Securing web service by automatic robot detection, in: Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference, 2006, pp. 23–23.
[9] V. Matyás Jr., Z. Riha, Toward reliable user authentication through biometrics, IEEE Secur. Priv. 1 (2003) 45–49.
[10] F. Bergadano, D. Gunetti, C. Picardi, User authentication through keystroke dynamics, ACM Trans. Inf. Syst. Secur. 5 (2002) 367–397.
[11] F. Monrose, A. Rubin, Authentication via keystroke dynamics, in: Proceedings of the 4th ACM Conference on Computer and Communications, Security, 1997, pp. 48–56.
[12] A.A.E. Ahmed, I. Traore, A new biometric technology based on mouse dynamics, IEEE Trans. Dependable Secur. Comput. 4 (2007) 165–179.
[13] M. Brown, S.J. Rogers, User identification via keystroke characteristics of typed names using neural networks, Int. J. Man-Mach. Stud. 39 (1993) 999–1014.
[14] L. Ballard, F. Monrose, D. Lopresti, Biometric authentication revisited: understanding the impact of wolves in sheep's clothing, in: Proceedings of the 15th Conference on USENIX Security Symposium, vol. 15, 2006.
[15] S. Gianvecchio, Z. Wu, M. Xie, H. Wang, 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.
[16] Blogbot 2.0 (2012 edition) by Incansoft <http://www.incansoft.com/IS0035.php> (accessed 08.03.2012).
[17] Ultimate Wordpress Comment Submitter <http://www.wordpresscommentspammer.com/> (accessed 08.03.2012).
[18] Autohotkey – Free Mouse and Keyboard Macro Program with Hotkeys <http://www.autohotkey.com/> (accessed 08.03.2012).
[19] Autoit, Automation and Scripting Language <http://www.autoitscript.com/site/autoit/> (accessed 08.03.2012).
[20] Autome – Automate Mouse and Keyboard Actions <http://www.asoftech.com/autome/> (accessed 08.03.2012).
[21] Global Mouse and Keyboard Library <http://www.codeproject.com/KB/system/globalmousekeyboardlib.aspx> (accessed 08.03.2012).

[22] Json, Javascript Object Notation <http://www.json.org/> (accessed 08.03.2012).
[23] C. Jackson, A. Bortz, D. Boneh, J.C. Mitchell, Protecting browser state from web privacy attacks, in: Proceedings of the 15th International Conference on World Wide Web, 2006, pp. 737–744.
[24] Virtual-Key Codes <http://msdn.microsoft.com/en-us/library/ms927178.aspx> (accessed 08.03.2012).
[25] S. Gianvecchio, H.Wang, Detecting covert timing channels: an entropy-based approach, in: Proceedings of the 2007 ACM CCS, Alexandria, VA, USA, 2007.
[26] Z. Chu, S. Gianvecchio, H. Wang, S. Jajodia, Who is tweeting on twitter: human, bot or cyborg?, in: Proceedings of the 2010 Annual Computer Security Applications Conference, Austin, TX, USA, 2010.
[27] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley-Interscience, New York, NY, USA, 2006.
[28] A. Porta, G. Baselli, D. Liberati, N. Montano, C. Cogliati, T. Gnecchi-Ruscone, A. Malliani, S. Cerutti, Measuring regularity by means of a corrected conditional entropy in sympathetic outflow, Biological Cybernetics 78.
[29] R. Kohavi, R. Quinlan, Decision tree discovery, in: In Handbook of Data Mining and Knowledge Discovery, University Press, 1999, pp. 267–276.
[30] J.R. Quinlan, Discovering Rules from Large Collections of Examples: A Case Study, Edinburgh University Press, 1979.
[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, SIGKDD Explor. Newsl. 11 (2009) 10–18.
[32] Attribute-relation file format (arff) <http://www.cs.waikato.ac.nz/ml/weka/arff.html> (accessed 08.03.2012).
[33] G. McLachlan, K. Do, C. Ambroise, Analyzing Microarray Gene Expression Data, Wiley, 2004.
[34] How much of the web actually work without javascript <http://tobyho.com/HowMuchoftheWebActuallyWorkWithoutJavascript> (accessed 08.03.2012).
[35] A study of internet users' cookie and javascript settings <http://smorgasbork.com/component/content/article/84-a-study-of-internet-users-cookie-and-javascript-settings> (accessed 08.03.2012).

Zi Chu received his Ph.D. degree in Computer Science from the College of William and Mary, Virginia, US in 2012. He is a Software Engineer with Twitter Inc., San Francisco, CA. His research interests include Web technologies, machine learning, data mining, Internet anomaly detection, security and privacy.

Steven Gianvecchio received his Ph.D. in Computer Science from the College of William and Mary in 2010. He is a Senior Scientist and Principal Investigator at the MITRE Corporation in McLean, VA. His research interests include networks, distributed systems, network monitoring, intrusion detection, traffic modeling, and covert channels.

Aaron Koehl is a PhD candidate at the College of William and Mary in Williamsburg, VA. He received his masters degree in systems engineering from the University of Virginia in 2005. He is an instructor of computer science and information systems at Christopher Newport University in Newport News, VA. His research interests include the mobile web, web application development tools, and networked enterprise systems.

Haining Wang received his Ph.D. in Computer Science and Engineering from the University of Michigan at Ann Arbor in 2003. He is an Associate Professor of Computer Science at the College of William and Mary, Williamsburg, VA. His research interests lie in the area of security, networking system, and distributed computing.

Sushil Jajodia is University Professor, BDM International Professor, and the director of Center for Secure Information Systems in the Volgenau School of Engineering at the George Mason University, Fairfax, Virginia. He received his PhD from the University of Oregon, Eugene. The scope of his current research interests encompasses information secrecy, privacy, integrity, and availability. He has authored or coauthored six books, edited 38 books and conference proceedings, and published more than 400 technical papers in the refereed journals and conference proceedings. He is also a holder of ten patents and has several patent applications pending. He has supervised 26 doctoral dissertations. Nine of these graduates hold tenured positions at US universities; four are NSF CAREER awardees and one is DoE Young Investigator awardee. Two additional students are tenured at foreign universities. Dr Jajodia received the 1996 IFIP TC 11 Kristian Beckman award, 2000 Volgenau School of Engineering Outstanding Research Faculty Award, 2008 ACM SIGSAC Outstanding Contributions Award, and 2011 IFIP WG 11.3 Outstanding Research Contributions Award. He was recognized for the most accepted papers at the 30th anniversary of the IEEE Symposium on Security and Privacy. His h-index is 71 and Erdos number is 2. The URL for his web page is http://csis.gmu.edu/jajodia.