Evasive Bots Masquerading as Human Beings on the Web

Jing Jin, Jeff Offutt
George Mason University
{*jjin3, offutt*}*@gmu.edu*

Nan Zheng
College of William and Mary
nzheng@cs.wm.edu

Feng Mao
Itron Inc
fengmao@acm.org

Aaron Koehl, Haining Wang
College of William and Mary
{*amkoeh, hnw*}*@cs.wm.edu*

***Abstract*—Web bots such as crawlers are widely used to automate various online tasks over the Internet. In addition to the conventional approach of human interactive proofs such as CAPTCHAs, a more recent approach of human observational proofs (HOP) has been developed to automatically distinguish web bots from human users. Its design rationale is that web bots behave intrinsically differently from human beings, allowing them to be detected. This paper escalates the battle against web bots by exploring the limits of current HOP-based bot detection systems. We develop an evasive web bot system based on human behavioral patterns. Then we prototype a general web bot framework and a set of flexible de-classifier plugins, primarily based on application-level event evasion. We further abstract and define a set of benchmarks for measuring our system's evasion performance on contemporary web applications, including social network sites. Our results show that the proposed evasive system can effectively mimic human behaviors and evade detectors by achieving high similarities between human users and evasive bots. This research provides a systematic way to explore the limitations of HOP-based web bot detectors.**

***Keywords*-Web security, bot, machine learning, human observation proofs**

I. Introduction

A *web bot* is a program that automatically and recursively traverses web sites, fulfilling one of many possible online tasks. Although automating online activity has improved the intelligence, performance, and efficiency of online systems, web bots also have significant negative impacts on Internet service providers and users. For example, in August 2012 it was reported that 80% of the clicks that a startup company paid for came from Facebook bots [1]. These bots automatically load pages and drive up advertisement costs, leading to financial losses for the online vendors.

Human interactive proofs (HIPs) such as CAPTCHAs are used in many web systems to distinguish bots from humans. CAPTCHAs require a user to pass challenge-response tests by either recognizing distorted characters or selecting images. However, the interactive requirement of CAPTCHAs makes them unsuitable for continuous and passive monitoring, and its usage is mainly limited to one-time authentication or registration. To make the bot detection transparent to online users, human observational proofs (HOPs) based approaches have also been developed [2], [3]. The design rationale behind HOP-based detection is that human behavior is more complex than bot behavior, and their interactions with the web application are also very different. These HOP-based systems are able to accurately distinguish bots from human users by identifying bot behaviors that differ intrinsically from human's behaviors, such as keystrokes and mouse movements.

From the perspective of web bot creators, some researchers have proposed evasive bots [4], [5]. The goal of evasive bots is to "poison" anomaly detectors and then evade the bot detection systems. Given the aggressive nature of web bots, we believe it is essential to understand the threat of evasion against HOP-based defense. In particular, the web bots that mimic human behavior patterns on the Web have not yet been systematically studied. This paper explores the limits of current HOP-based bot detection systems by creating a new evasive bot system that masquerades as human beings on the Web. Specifically, we characterize the existing HOP-based web bot detectors and develop an evasion framework based on human behavior patterns. Instead of subverting a specific detection system, the major goal of this study is to provide a systematic approach to evaluate and explore the limits of current HOP-based detection systems, and motivate online business to develop more advanced bot detectors.

This paper makes three contributions. First, we characterize web bot detection observation proofs that use web application events, with the focus on the different events and their statistics associated with the activities of bots and human clients, such as timing and location. Second, we propose a generative approach to build an evasive web bot via adaptive human behavior learning. The proposed web bot can mimic human behaviors to hide its activities within legitimate user events; in other words, de-classifying bot-like traffic. Third, we provide a practical framework for evasive web bot design and implementation, which helps researchers and engineers to explore the limitations of existing bot detectors.

We have implemented a prototype of the proposed evasive bots, and validated the power of our evasive bots by measuring the *similarity* on various metrics from a set of selected web representative applications. In our experiments, we redefine the Kullback-Leibler (KL) distance formula [6] to evaluate the effectiveness of the proposed evasive model and algorithm. Our experimental results show that the proposed evasive system can effectively mimic human

behaviors and achieve high similarities between human users and evasive bots. Note that the purpose of our experimental evaluation is not to show how successful our approach can bypass a specific HOP-based detector, but to demonstrate how close our proposed evasive bots are to normal human users, which will ensure our approach to evade any existing HOP-based detectors. To capture our proposed evasive bots, future HOP-based detectors must introduce new features for bot detection. Otherwise, even if future detectors further tune up their classifiers, the detection of evasive bots will result in many false positives, due to the very close behavioral similarity between our evasive bots and human users, making these fine-tuned detectors useless.

The remainder of this paper is organized as follows. Section II characterizes bot detection, focusing on application events. Section III proposes a generative approach for evading bot detection. Section IV presents our framework to develop evasive web bots. Section V evaluates the evasive capability of the proposed evasive bots by testing them in several detection benchmarks. Section VI reviews the literature of bot detection and evasion, and finally Section VII concludes the paper.

II. Characterization of Behavior-Based Web Bot Detection

Web browsers capture user action behaviors by generating events on the web application such as keystrokes, mouse motions, and clicks. The application events often trigger traffic events that can be observed on the server. As Figure 1 shows, detectors can utilize both application events and traffic events for anomaly detection by analyzing user action events and classifying bot features in network traffic characteristics. Since the traffic events are driven by the application events, our goal is to study human input event patterns and generalize behavior-based web bot detection from the perspective of application events. We also discuss traffic events but only for the purpose of filtering noise during user action collection. We start with simple boolean events, which identify bots by capturing certain unexpected or expected human behaviors. Then, we examine a more complicated scenario where bot detection relies on various statistics of event metrics.

A. Detection Abstraction

Advanced bot detectors use machine learning to identify important features. Let R denote the set of bot actions, and H denote the set of human actions. We define human input X, traffic events Y, and the decision maker function F, also called the *binary classifier*. Detectors learn and build the classifier function $F : (X, Y) \rightarrow \{0, 1\}$ from a training set of (X_h, Y_h) from H, compared with (X_r, Y_r) generated from R. The classification process is formulated as $Z = F(X, Y)$, where Z represents the output of the classifier, $Z \in \{0, 1\}$.

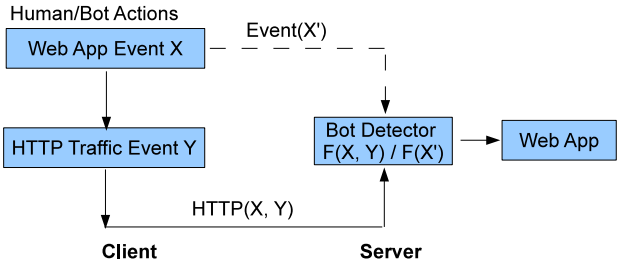

Figure 1. The behavior-based detection process

We assume that X is a set of all actions of a web user. We also assume the detector is able to collect user events in a timely manner. From the detector's perspective, the user input is taken in two different scenarios as shown in Figure 1. The first scenario is when human input events generate network traffic. In this case, X and Y are dependent, and we define the formula to be $F : Z = F(X, Y)$. For example, a click on a web link generates traffic by downloading a web page. The second scenario is when a user action X' such as a mouse movement without a click does not generate any network traffic. We can simplify the formula to be $F : Z = F(X')$ in this scenario. In Figure 1, the user input of the first scenario is shown with a solid line, while that of the second scenario is presented as a dotted line.

B. Application Event-Driven Detection

A web browser on the client notifies web applications of changes by generating user input events, such as keystrokes, mouse movements, and clicks on buttons and links. These application events are collected and sent to the server. The detection engine on the server notifies the web application administrator when detecting abnormal web bot activities.

Boolean events: We summarize some common event-driven detection features in Table I. The simplest events are boolean events, as a special case of event statistics, which represent expected or unexpected presence. These event-driven detectors perform online detection on the server. Most detectors try to identify bots in real time. This set of detection techniques is usually combined with specific web implementation technologies such as client-side JavaScript and form tracking. We formalize categories of keyboard/mouse, random elements, and hidden elements as a 0 and 1 boolean pattern, because the user input X to the classification function F is also a boolean variable. The classification function F is straightforward, either $F : Z = X$ or $F : Z = \neg X$. This kind of detection takes advantage of the prior knowledge of differences between real human input actions and bot input actions to the web interface.

Events Statistics: The application event-driven detection also uses statistical features observed from human inputs on the client side. These detection approaches are based on the observation that bots' behaviors vary less than humans'. For example, a bot may periodically repeat one

Human Action	App Event Variable X: Trigger —	Timing —	Location/Obj —	Detection Function F	Semantic
mouseclick	X	-	-	$Z = X$	movement
mouseover	X	-	-	$Z = \neg X$	movement
keystroke	-	-	X hidden form	$Z = \neg X$	abnormal
mouseclick	-	-	X hidden link	$Z = \neg X$	abnormal
mouseclick	-	-	X Random Element	$Z = X$	integrity
mouseclick	-	X	-	$Z = F_{dist}(X)$	behavior
mouseclick	X_1	X_2	-	$Z = F_{rate}(X)$	behavior
mousemove	-	-	X	$Z = F_{dist}(X)$	behavior

Table I
WEB APPLICATION EVENT-DRIVEN DETECTION

Event	Metrics: count	timing	location
click		x	x
move	x		x
keystroke	x	x	

Table II
METRICS SELECTED DETECTION/EVASION ANALYSIS

action or periodically trigger a sequence of actions, while human actions follow a normal or Pareto distribution. These kinds of detectors usually attempt to detect bots in real time with simple implementations. In this scenario, widely used distribution models like uniform, Pareto or Weibull distributions [7] are assumed. More sophisticated detectors based on the estimation of more complex model parameters learned from human behaviors can distinguish bots from humans.

III. GENERATIVE EVASION

This section describes how bots masquerade as human users. We select event-based metrics from the detector's perspective, and demonstrate how bots can mimic human action events in timing and location. Further we define similarity metrics according to revised symmetric KL distance measurement, assuming the HOP detector would also use KL distance [8] to distinguish bots and humans. We also build a pool of candidate models to best fit human behaviors, followed by an algorithm that selects the right model to fit user actions to create evasive bot actions.

Target Behavior Metrics

We select a set of application independent events such as keystrokes, mouse clicks and movement counts, and measure timing and location, as listed in Table II. Timing represents inter-event timing between two subsequent events, while location demonstrates the mouse click positions and movement angle information. These are used for statistical behavior analysis. We also count how many times the move and keystroke events appear.

Boolean Behavior: Note that boolean events can be integrated into the statistics distribution-based detection as a special case. For example, a click on a hidden link is a single event. The distribution statistic is a simple 1/0 constant. The statistic is obtained either from prior knowledge by analyzing the code or by adversarial classification. The boolean behavior events are usually constructed as a set.

Behavior Statistics: For event timing, we apply statistics instead of a single value to characterize its behavior. We list all the behaviors in Table II. We map each histogram into a point in a space and measure the similarity of two distributions by the distance in the space. This kind of distance can be measured in many ways. We create a new distance formula in Section III-A by adapting the KL formula.

Timing-based Events: We call the timing events from a session *micro events*. To be more specific, micro events come from user actions on a client that generate network round-trip times or HTTP requests. We are particularly interested in inter-event timing in micro events. Since the round-trip times are relatively small and exist for both bots and humans, we can treat them as random noise and calculate idle and inter-event times equivalently. Naive bots usually generate a large number of events almost constantly, leaving clear timing patterns for detection. Our bot framework allows bots to insert idle time to mimic human behaviors.

Location-based Events: The two movement events are mouse clicks and movement, as they carry the basic information about human actions. Suppose we have a sequence of click actions on locations $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$. The movement can be characterized as a movement vector between two locations. The vectors are defined as $(x_1, y_1) \rightarrow (x_2, y_2)$, $(x_2, y_2) \rightarrow (x_3, y_3)$, ... $(x_{n-1}, y_{n-1}) \rightarrow (x_n, y_n)$. The correlation between the movements are the angles of two contiguous vectors. The angles are defined by consecutive pairs of vectors: $\theta = \arctan((y_2 - y_1)/(x_2 - x_1))$. We map the angle from the range $[-\pi, \pi]$ to $[0, 360]$ to get a more intuitive explanation.

A. Similarity Measurement

Formally, we treat a contiguous distribution as a discrete histogram calculated from the observation data, which is normalized into n buckets. We then map each histogram into a point in an n-dimensional space. The similarity of two distributions is measured by the distance between two points

in an n-dimensional space. The KL formula in equation (1) can quantitatively compare different distributions or histograms. H_1 and H_2 stand for two normalized histograms. $H(i)$ means the normalized ith bucket value in the histogram H, where $\sum_i H(i) = 1$.

$$D_{KL}(H_1, H_2) = \sum_{i=1}^{n} H_1(i) \log \frac{H_1(i)}{H_2(i)} \quad (1)$$

Intuitively, KL divergence turns the difference in each bucket into a real number with two directions, negative and positive, depending on their relative values, and then weights real numbers with H_1. However, this is not a distance metric on the space of a probability distribution, because it is not symmetric; that is $D(H_1, H_2) \neq D(H_2, H_1)$. We define a new distance measurement based on the KL formula equation (2). The new definition is a distance metric, and is symmetric, that is $D(H_1, H_2) = D(H_2, H_1)$. It removes the direction on the original definition in each bucket and averages the bi-directional differences. The bigger the value of the distance is, the larger the difference there is between the behaviors. Practically, the different distributions detected on the server generate larger KL distance values for the classifier to identify the bots.

$$\begin{aligned} D_{SKL}(H_1, H_2) &= \frac{1}{2}\Big(\sum_{i=1}^{n} H_1(i) \log \frac{min(H_1(i), H_2(i))}{max(H_1(i), H_2(i))} \\ &+ \sum_{i=1}^{n} H_2(i) \log \frac{min(H_1(i), H_2(i))}{max(H_1(i), H_2(i))}\Big) \\ &= \sum_{i=1}^{n} \frac{1}{2}(H_1(i) + H_2(i)) \left| \log \frac{H_1(i)}{H_2(i)} \right| \quad (2) \end{aligned}$$

Behavior Consistency Degree: The *behavior consistency degree* (CD) can quantitatively evaluate how a particular behavior varies across users of a web application. It indicates how hard it is to capture the user behavior for our evasive model.

Formally, assume we have a set of user behavior histograms $U = \{u_1, u_2, ..., u_n\}$ such that:

$$CD(U) = \frac{\min D_{SKL}(c, U)}{\max D_{SKL}(c, U)} \times mean(U, c) \quad (3)$$

where $c = mean(U)$.

Intuitively, if we treat each user histogram vector as a point in the space, CD represents how the points are spread in that space. A large CD means irregular behavior that is difficult to capture. The first factor represents how points are spread in $[0, 1]$; the second part is the average distance between the center and user points in the range of $[0, \infty]$.

Algorithm 1: Model selection algorithm

Data: A sequence of human user data $U =< u_1, u_2, ... >$, buffer size W, retired data size α

Result: A sequence of the best model and its parameter estimation $< M, P >$

```
W = {u_1, u_2, ..., u_W} ;
while U ≠ ∅ do
    buffer the user data into set W ;
    if reach the buffer size W then
        C = (C − {c_1, c_2, ..., c_α}) ∪ W ;
        calculate the center of the user data c;
        distance = ∞ ;
        for m ∈ ModelPool do
            fit c with m ;
            generate c_m;
            if D_SKL(c_m, c) < distance then
                M = m;
                P = estimated parameters;
                distance = D_SKL(c_m, c) ;
            end
        end
        U = U − W ;
    end
end
```

B. Evasion Model Fit and Selection

Model selection is the problem of choosing among different mathematical models that all try to describe the same data set. Since no single model can fit all user behaviors, we build a pool of candidate models and choose the best matching model dynamically. The model can be chosen in several ways, including optimizing some measure of goodness of fit, model averaging, and cross validating. Our goal is to optimize D_{SKL}, the measurement of goodness distribution fit. Instead of statically selecting the best model, our generative approach selects the model dynamically by evaluating different models simultaneously. We design an algorithm that looks for the best distribution fit for a sequence of time windows. We smooth the model selection with a mixture of old and new data to fit the models. Algorithm 1 shows the procedure.

Although we could use a history of human events to build a prediction model for each particular input event X, we decide to reuse existing models to build a model pool for easy and efficient implementation. Previous studies indicate that well studied models like the Fitts' law equation [9], [10] could fit the human behavior well. Our model pool has seven models.

We use the common method of maximum likelihood to find the local optimized fit. For example, given the Pareto model selected from the model pool, we use its likelihood

function to estimate the parameters α and t_m. Given an n-history sample of event timing intervals $\{t_1, t_2, ...t_n\}$, the logarithmic likelihood function is:

$$\log(L(\alpha, t_m)) = n\log(\alpha) + n\alpha\log(t_m) - (\alpha+1)\sum_{i=1}^{n}\log(t_i) \tag{4}$$

To maximize the likelihood function, we use the estimation $t_m = min(t_1, t_2, ...t_n)$. When

$$\frac{d}{d\alpha}\log(L(\alpha, t_m)) = 0 \tag{5}$$

and

$$\alpha = \frac{n}{\sum_{i=1}^{n}(\log(t_i) - \log(t_m))} \tag{6}$$

the maximum likelihood function can reach its maximum value.

In addition, the selection algorithm continuously re-estimates the parameters when the bot receives event timing patterns from the users. Thus, our evasive bot can gain knowledge from history and integrate it into future data. This has two advantages. First, this design can adapt to the target server automatically when there are changes on the server that could cause the timing pattern to change. Second, the adaptive algorithm can adjust parameters of a model and continue to deceive the detector.

IV. An Evasive Framework

We implemented a prototype of the evasive bot system that supports multiple web agents. Conceptually, the framework is a continuous learning system. Human actions trigger events that train a generative engine, so the web bots can later mimic the human behaviors. The system design is summarized in Figure 2. The *human data flow* is shown with dashed arrows and the *web bot data flow* is shown with solid arrows.

Human data flow represents human data being collected and pre-processed. Information is retrieved from human generated events and traffic, and then saved in logs. This framework records human data with a recorder on the client and an HTTP proxy, and then the human data collected becomes history data for the bot engine to learn human-like behavior. As shown in *human data flow* in Figure 2, human events and traffic behaviors are first recorded in logs on the client and through HTTP proxies. As logs get pre-processed, the generative engine studies the human behavior from the log and then supports later use by the event generator.

Bot data flow starts with a bot action specification whose most important component is the event generator. The bot action spec defines all types of actions, including mouse clicks and keystrokes. The event generator selects specifications from a set of bot actions, and then makes decisions about how to generate the bot events. The event generator applies the Selenium web driver API, which has full control of web browsers. This allows it to take advantage of any facilities offered by the native platform. Since the platform supports native events in Windows or Linux, we can easily send a mouse click event with the element's coordinates relative to the border of the screen at the OS level. Both keystrokes and mouse events use the native OS API. We implemented the event generator as a $3,000$ line Java program that sends events to web applications.

Web Agent: The web agent runs the web application with JavaScript. It captures application events and passes them to the event logger. We use web browser plugins to monitor and control web application events in JavaScript.

HTTP proxy: The HTTP proxy captures detailed information about HTTP transactions and passes the information to the event logger. As any web event can trigger an HTTP request, it contains more detailed information such as timing information and element XPaths. We did not capture application semantics embedded in the payload, leaving them to other data capture methods.

Event Logger: This component obtains human behavior data from different data sources and performs a pre-processing step on the raw data before feeding it to the generative engine. The client-side web browser, OS, and HTTP Proxy capture different aspects of user behavior data. The JavaScript running inside the web browser can capture application specific events. We take advantage of tools like Record User Interaction (RUI) [11] at the OS level to help capture device events such as mouse clicks and movements. The HTTP proxy captures and records more detailed information about HTTP transactions that are associated with the human. Before the event logger stores the human behavior data into the log database, it performs a pre-processing step on the three data sources. Our implementation aligns the data according to timing and filters out unnecessary data before compression.

Generative Engine: The generative engine queries the log database for the user behavior data. It analyzes the collected human behavior data and generates a suitable behavior model, then passes it to the event generator. For the boolean function, the generative engine adds a rule for each boolean fact observer from the human, such as a requirement to send a mouse movement event. For the behavior statistics, the generative engine extracts the human behavior and outputs a generative model by selecting the best fit. For example, it passes a timing model to perform bot actions, which specifies the delay time among bot actions that trigger events.

Event Generator: The event generator generates events according the bot action specification and resolves the constraints by querying the generative engine for information such as timing and location. The event generator first builds a data structure for elements from Table III using the web driver's API. The web driver identifies the web element

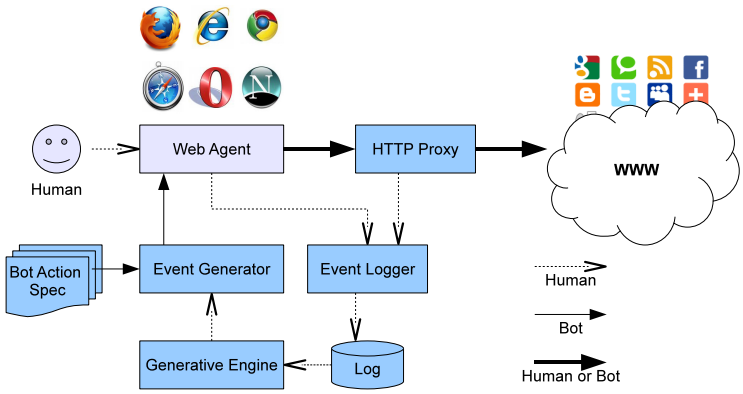

Figure 2. Bot system framework

Input Events	Web Element XPath (or ID)	Command calls
click	//a[text()='some url'/@href	navigate()
`mouseover`	//div[@id='dropdownlist]/	hover()
`select`	//a[text()=' second']	setSelected()
`type`	/html/body/div/table/input	sendKeys()
`click`	//button@value='submit'	submit()

Table III
ACTION SPECS EXAMPLE

according to the unique XPath of the web element. Table III shows mappings between events and Java calls. In addition, the event log contains an event timing log and an event sequence log. For example, the event module determines a time sequence from the human event timing log. The action interarrival times, $t = (t_1, t_2, ..., t_i)$, by default are constant as a bot. But our goal is to create user events that follow human-like events, so we define several models using distributions such as normal and Pareto. Finally, the event generator component generates a web bot action sequence and sends commands to the web agents.

Bot Action Spec: The bot action spec defines a sequence of event commands and evasions. These actions are mapped to several Java calls in the event generator. Each event command is a triple $<action, element\ ID, bot\ command>$. Table III describes the primary elements in the specification. The timing and movement patterns are encoded in the sequence as "*nop*" actions. Those behavior events are generated during runtime according to the model from the generative engine.

V. EVALUATION

This section empirically evaluates the evasive capability of our evasive bot framework. We first clarify the challenges of selecting benchmarks and creating workloads. Then we present results from testing evasive bots using different detection behaviors. As mentioned before, the objective of our evaluation is not to demonstrate how successful our framework can bypass a specific HOP-based detector, but to quantify the similarities between the behaviors of our evasive bots and those of normal human users. The behavioral closeness between our proposed bots and human users will enable our approach to evade existing HOP-based detectors.

A. Benchmark Set Definition

Evading all possible behavior metrics that detection mechanisms could use is quite challenging. First, it is difficult to fully understand all the detection implementations and models deployed on commercial web servers, because the source code is usually not available and specifications are not public. Second, detectors usually rely on one or several metrics that are specific to web applications, possibly both signature-based metrics and anomaly-based metrics. It is hard to know what behavior metrics are used by the production bot detectors, and different companies might use different metrics. Finally, even if we ambitiously implement a long vector of metrics for evasion, the complexity of the detection mechanisms still make it difficult to evaluate the bots' ability to evade them.

Our strategy is to select some typical metrics from a wide range of commonly used web applications. We measured the event timing in each session for five widely used web applications. The web applications were chosen to cover most typical web uses. We logged and measured four types of web applications: (1) a web email application, which included logging in, and composing, sending and retrieving emails; (2) two social networking sites to update status messages, communicate with friends, etc; (3) a web news system; and (4) a blog application, which has one update and multiple read patterns.

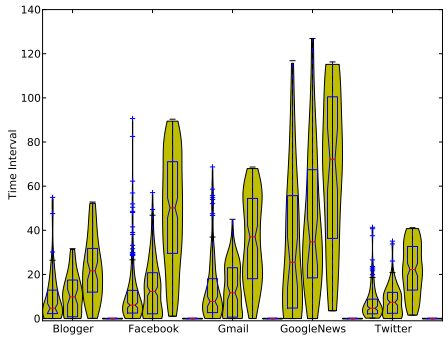

Figure 3. Inter-click timing in seconds for human and proposed evasive bots. From left to right in each group, they are human data, a normal distribution model, and a uniform distribution model.

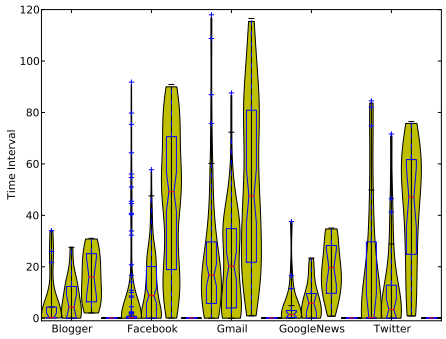

Figure 4. Inter-keystroke timing in seconds for human and proposed evasive bots. From left to right in each group, they are human data, a normal distribution model, and a uniform distribution model.

B. Data Collection

We selected five web applications: Gmail, Facebook, Twitter, Google News, and Blogger. We collected human data by inviting 35 people to participate in 50 experiments to capture their real-time behaviors. The users' ages range from 18 to 50 and they are located in different regions in US; 15 on the west coast and 20 on the east coast. We anonymized all users' personal information to protect their privacy. Our framework captured both application events and traffic events including keyboard, mouse movements, and mouse clicks.

C. Experimental Setup

We built our evasive web bot framework on a mainstream desktop configured with an Intel core Dual CPU 2.40 GHz and 4GB RAM. The operating system was Windows 2008. We used Mozilla Firefox $3.5.X$ as our primary agent. It can be configured to represent three common agents, and can be extended to mobile web agents.

We implemented the bot event generator in Java 1.6. The event generator controls the web agent or browser through a plugin. Our experiments were conducted in a high speed Internet service environment, sending traffic to a variety of web servers. There are no special network restrictions for our framework, and it can be scaled to a large number of virtual machines running on clusters or a cloud to gain extra computation power.

We also built an Ubuntu 9.10 Virtual Machine running an Apache 6.0 HTTP server to construct a synthetic web site to evaluate boolean events.

D. Evaluation of Evasion Capabilities

This section presents data from an experimental demonstration of how our framework can prevent web bots from being detected by masquerading as humans. The features used are common to web bot detectors.

1) Evaluation Approach: One approach to measuring the evasive capability of our bots framework is to test bot detection in the presence of our evasive attacks. If we treat bot detection as a special case of an intrusion detection system, it is easier to understand the challenges of creating an appropriate workload for testing. First, most anomaly detectors require a large amount of training data to build the classification criteria. This leaves the question of how much data is enough. Second, there are no clear guidelines for how to select representative human user inputs. Web applications have become more complex over time, so historical data may not always be applicable. Third, randomness is introduced into the execution environment by the server workload, network traffic, the security protocol, asynchronous communication from Ajax, and user preferences. The Web also introduces new dynamic functionalities that increase the challenge of quantifying the workload [12]. Finally, no standard test methodology is available [13].

These challenges make it difficult to develop a general evaluation method for detectors that is repeatable and comparable across many web applications. Furthermore, the measurement metrics could be tightly coupled with the implementation of certain specific detectors that are not publicly available to us.

We avoid the problems above by taking different approach. In particular, we quantify and compare the similarities between human and evasive bot behaviors. The evasive bot behavior is generated according to real human behavior. It is reasonable to quantify the similarity by calculating the distance between two vectors of behavior features. The more similar to human behavior, the more evasive our bot is considered to be. For boolean events, we evaluated the evasion detection results, as we have full control and understanding of the detection systems in the synthetic web sites.

2) Web Application Boolean Events: Our boolean events are automatically generated according to the logged human

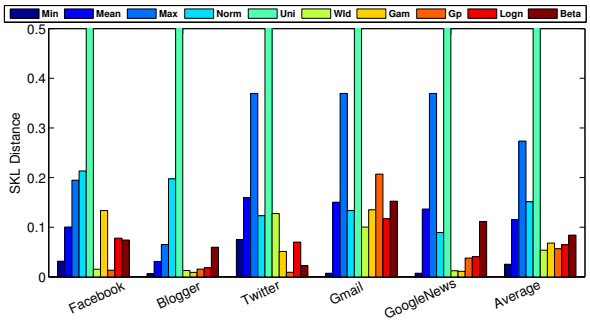

Figure 5. D_{SKL} for inter-events timing of click actions across different models. From left to right in each group, the first three are min, max and mean of all distances, and collectively work as a baseline. The other seven are distance fittings from normal, uniform, Weibull, gamma, Gp, Logn and beta distributions.

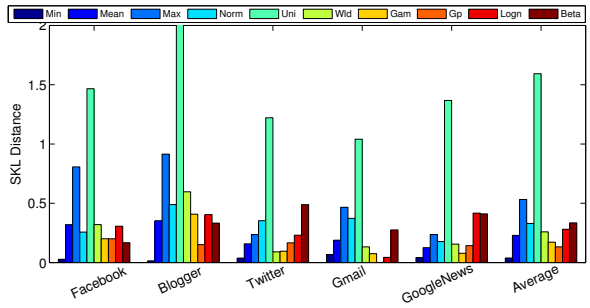

Figure 6. D_{SKL} for inter-events timing of keystroke actions across different models as in Figure 5.

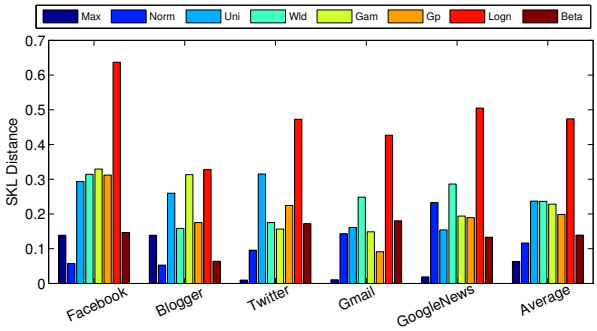

Figure 7. D_{SKL} for angles of mouse movement for different models. From left to right in each group, the first is the max of all distances, and collectively work as a baseline. The other seven are distance fittings from normal, uniform, Weibull, gamma, Gp, Logn and beta distributions.

events. We mark "Pass" if the bot was not detected and "Fail" if the detection module raised an alarm. Table IV summarizes the detection and evasive results. Mouse movements and keyboard native I/O events are easy to evade. One limitation of the evasive bot is that it has limited coverage for web testing as it only follows human's footprints.

3) Web Application Event Timing: Event timing is defined as time intervals between events that are triggered by actions. Actions are those common tasks that both humans and bots need to perform, such as mouse clicks and key

User Actions	Results
Mouse over/movement	Pass
Key Strokes	Pass
Random Element	Pass

Table IV
EVASIVE CAPABILITY FOR BOOLEAN EVENTS

strokes. We are particularly interested in the inter-events time of various web applications with different usage patterns, as presented in Figure 3. For each timing sequence action record, we fit the distribution model via our candidate models to find the best fit. Then, our generative engine generates a timing interval sequence according to the selected model to evade the events triggered by bot actions.

We group each application into "violin plots" in Figures 3 and 4. A violin plot is a combination of a box plot and a kernel density plot. It summarizes the data distributions in five numbers: the smallest sample observation, the lower quartile, the median, the upper quartile, and the largest sample maximum. As our generative approach tries to maximize the distribution similarity, the violin plot is an intuitive way to graphically depict groups of numerical statistic data similarity. For each application in the figures, from left to right, the plots represent the human data and then the two generative data from the fitting models. The first fitting model is a normal distribution, which largely captures random human activity. The other is a uniform distribution. We map the similarity of two distributions from intuitive pictures to a number, the D_{SKL}. A summary of the inter-event timing interval D_{SKL} across all candidate models is illustrated in Figures 5 and 6.

To estimate the consistent degree of human behavior, we use the statistics represented by the first three bars of min, max, and mean from Figures 5 and 6 for each application. We first locate the center point from the distribution of all the human data across all users, and then locate the central point from each human user data. Finally, we calculate the distance between the overall center point and the central point of each individual user data. These three distances of min, max, and mean will be also useful for further detectors to define their own detection thresholds.

Then we measure distance fittings from normal, uniform, Weibull, gamma, Gp, Logn and beta distributions. We observe that the different distributions perform significantly differently, no single model dominates, and each model displays different performances across different applications. We confirm that our generative engine needs to select the models dynamically. Current web bots usually use the best effort model or simple uniform behavior statistics.

4) Web Movement Events: We further compare the evasive fitting models to human movement models. The *event movement* is defined as the angle and distance between two continuous actions. Because distance correlates with the

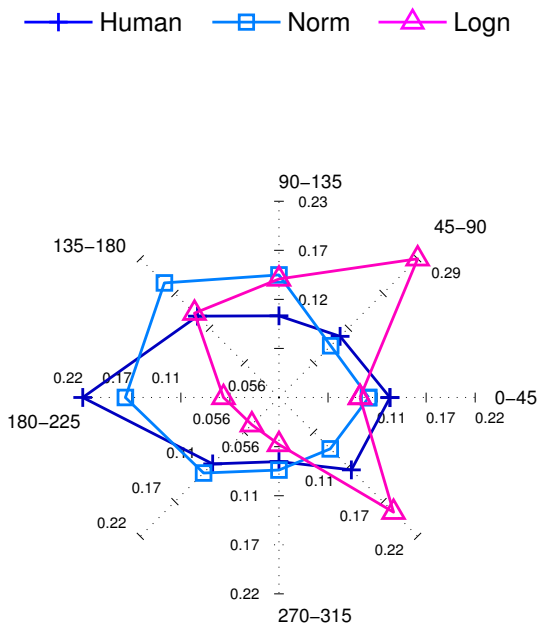

Figure 8. Facebook mouse clicks and movement angles

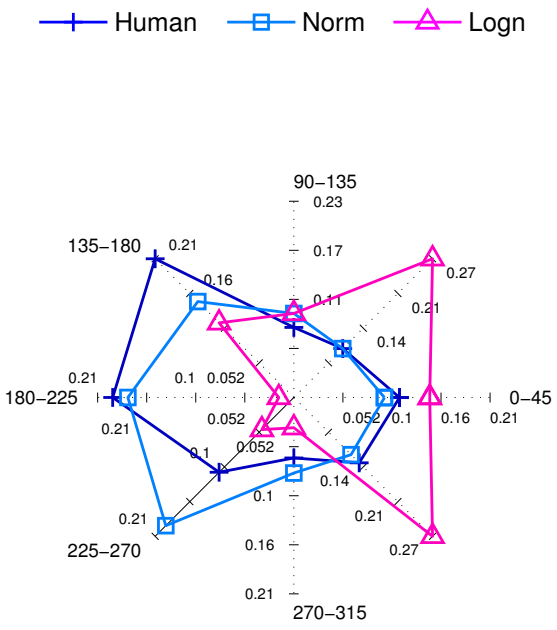

Figure 9. Blogger mouse clicks and movement angles

timing behavior, here we only focus on the angles.

We measure the mouse movement in each session for the five web application benchmarks used in Section V-D3. Because the distribution is in a closed area, we map the value from $[-\pi, \pi]$ to $[0, 360]$ to display the distribution similarity intuitively. We evaluate the similarity between human and different generated movement behavior in spider graphs, as shown in Figures 8 through 12. Each graph has one line for the human behavior and contains up to seven lines for each generative model. To display the similarity clearly, we only draw the two models with the largest and smallest distances.

The next step is to compare the bots' behaviors with the humans'. First we convert the closed paths in the spider graphs to a number defined by the D_{SKL}. For each web application, we calculate the D_{SKL} between human data and bot generated data via different fitting models. Because the user distance is relatively small compared to the fitting model distances, we only show the maximum user distance to the human behavior distribution.

A summary of the inter-event movement's D_{SKL} across all candidate models is displayed in Figure 7. Unlike the timing, the user movement behaviors are very consistent and the threshold can be very strict. From these data, it appears that no bot generative fitting model can consistently avoid triggering an alarm. But in some cases, such as the Facebook and Blogger applications, the generative model can hide the bot behavior quite well.

5) Fitting and Model Selection: Our evasive model fitting and selection employ real user samples to gradually rebuild its training set. We use the Gmail web application as an example to demonstrate our dynamic model selection procedure to identify the best model to mimic humans. Figure 13 shows data from the procedure to mimic the human timing. In this case, the sample window size is set to one. With the human data, the framework takes the behavior data and fits it with the seven models in our model pool. Based on the distance measurement, the framework selects the best model for the observation window. The upper subgraph tracks the selection movement among the models and the lower subgraph shows the minimum distance across the fitting results using the seven models. We plot a similar graph for the movement minimum in Figure 14. Interestingly, we can see that the best model could change among different users, primarily because human behaviors may differ. Different web application's functionalities can also cause subtle differences. In general, our framework captures human behaviors to evade the bot detector, and adjusts its model to mimic diverse human usage and rebuild the training set for a better model.

E. Defense Against Evasive Bots

Although we have proposed an evasive bot and provided a systematic approach to exploit the limitations of HOP-based detectors, our ultimate goal is to improve the security of

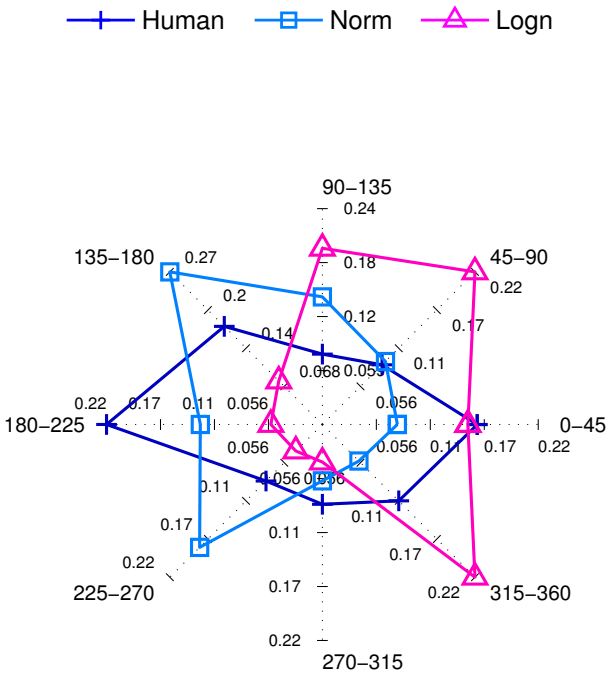

Figure 10. Twitter mouse clicks and movement angles

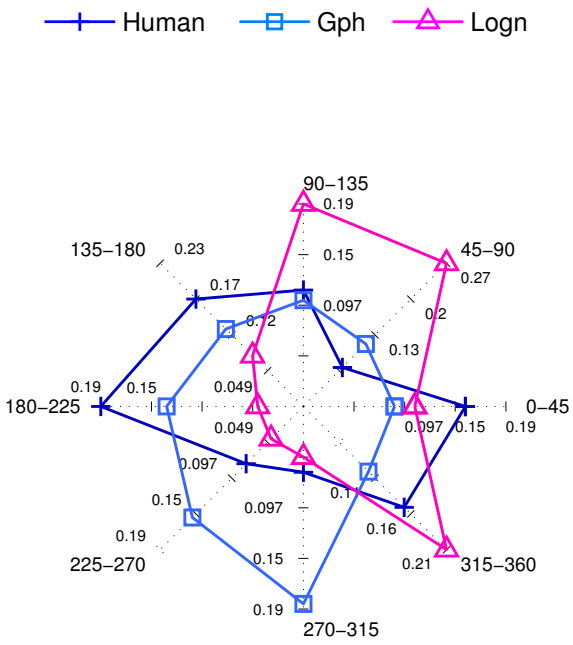

Figure 11. Gmail mouse clicks and movement angles

online business by developing more advanced bot detectors. To defend against evasive bots, future detectors could take application context and user intent into account for bot detection. In the context of online applications, human users can easily perceive the application feedback and interact with the applications in a logical and sensible manner. But for bots, it will be challenging and computationally expensive to continually analyze the application context and then adjust their behaviors with the changing context in a timely manner. Moreover, the intent of a human user is very different from that of a bot. For example, in a web browsing scenario, a human user clicks on several links to read multiple web pages, while bots may click on the same link several times to abuse pay per click. The evasive bot can inject mouse movement actions and idle time between clicks to mimic a human, but it cannot fully hide the final results and its intent. Finally, the user behavior consistency degree we defined before could also be used to detect evasive bots, because evasive bots usually have a higher consistency degree than human users.

VI. Related Work

Web bots automate various online tasks, serving good [14] or innocuous purposes (e.g., search engine bots), as well as malicious purposes (spam bots, vote bots, and game bots) To tackle the ever increasing threats from malicious bots, several serious research projects have tried to differentiate human users from bots. Traditional approaches such as CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) [15] differentiate humans and bots based on active user interactions (users are prompted explicitly for a test), and are generally known as human interactive proofs (HIPs). CAPTCHAs have been found to be an effective challenge response Turing test, and are deployed in many online services (including online account registration, online voting, and message posting). However, it is not suitable for continuous monitoring, as users must wait additional time ($\sim$10 seconds) to be verified [16]. Such frequent interruptions can cause an intolerable usability burden on users. But if users are not continuously monitored, human attackers can pass the one-time test and then let web bots take over. Recent studies have found that web bots often solve CAPTCHAs by using powerful optical character recognition techniques or cheap human labor [17], further lowering barriers for attackers to evade detection.

Unlike HIPs, human observational proofs (HOPs) take a non-interactive approach, passively monitoring input behaviors. Thus it is suitable for continuous bot detection. Measurements have found strong evidence for complexity

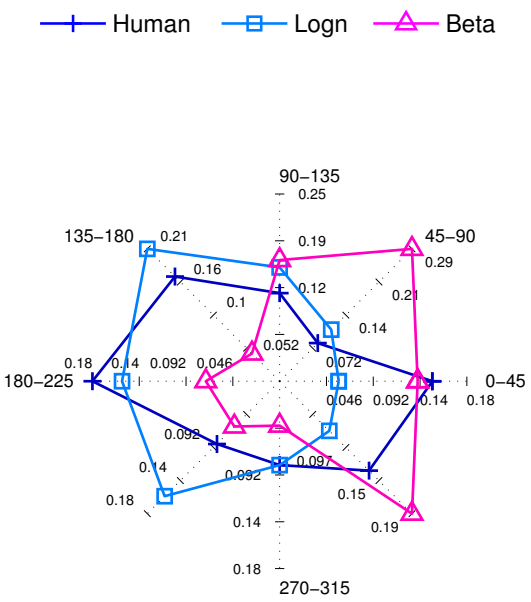

Figure 12. GoogleNews mouse clicks and movement angles

and irregularity of human behaviors that are rarely found in existing web bots. Gianvecchio et al. exploited these intrinsic differences to accurately detect chat bots (in online chatting) [2], as well as game bots (in online games) [3]. Critical behavioral differences from humans are also found in twitter and blog bots [18], [19], allowing detectors to use advanced machine learning techniques to detect malicious bots in Twitter and blogs.

Spam posting behaviors also differ from legitimate postings, and can be leveraged to detect those bots. Thomas et al. [20] presented Monarch, which can crawl and analyze URLs submitted to servers in real time, and determines which URLs lead to spams. Gao et al. [21] proposed an effective technique to filter out spam campaigns in online social networks (OSNs). In their approach, OSN spam messages are reconstructed into *campaigns*, and can be detected in real time with high accuracy and efficiency. Jacob et al. [22] presented PUBCRAWL to detect web crawlers based on the observation that the traffic generated by crawlers is very different from the normal user traffic.

There were various evasive approaches against web bot detectors. Spam filters can be evaded by poisoning training data with spam messages [4]. Bots can also use anonymous networks [23] to hide their identity so that it is less likely for the detectors to trace the source of web bots. In contrast, our attack focuses on web bots that mimic human behavior in a variety of ways, and applies to many more web applications. Additionally, in our attacks, compromised routers are not needed, as in Google Clickbot.A [5]. Evasion attack also occurs in evading biometrics-based authentication systems. Ballard et al. [24] presented a generative attack model by synthesizing handwriting automatically, which has been effective in evading existing handwriting based user authentication.

The technique of automatically mimicking human interactions with Ajax-based applications has been proposed in the AjaxTracker tool [25]. However, the goal of AjaxTracker is to generate workload and monitor network behaviors, instead of launching an evasive attack as in our work. Moreover, we provide a more flexible and extensible bot framework that directly controls a web browser.

VII. Conclusions

Web bots have been widely deployed to perform tasks on the Web in an automatic fashion. However, the behaviors of existing web bots are intrinsically different from those of human beings, which allow them to be detected by HOP-based bot detectors. This paper proposes a generative approach to build an evasive web bot. Based on the generative evasion, we have prototyped a generic web bot framework that mimics human behaviors on the Web. The framework enables bots to generate events at the application level for evading bot detectors. We have abstracted and defined a set of benchmarks and metrics to measure our system's evasion performance. Our experimental results demonstrate that our evasive bots can achieve high human likelihood against bot detection. This evasive bot framework can now be used to evaluate existing bot detection algorithms and help develop more advanced bot detectors.

Acknowledgments

We would like to thank 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] rt.com, "Startup dumps Facebook, alleging 80% of clicks came from bots," http://rt.com/news/facebook-social-media-startup-scam-ads-554/.

[2] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, "Measurement and classification of humans and bots in internet chat," in *Proceedings of the 17th USENIX Symposium on Security*, 2008.

[3] S. Gianvecchio, Z. Wu, M. Xie, and 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*, 2009.

[4] B. Rubinstein, B. Nelson, L. Huang, and A. J. etc, "ANTIDOTE: Understanding and defending against poisoning of anomaly detectors," in *Proceedings of Internet Measurement Conference*, 2009.

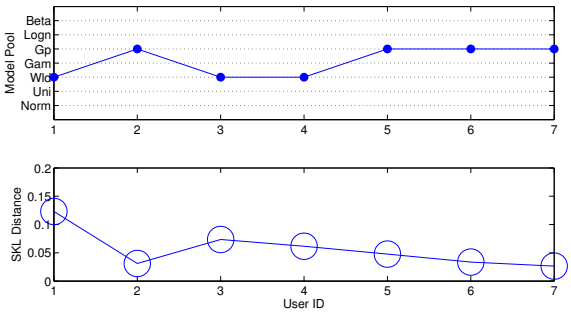

Figure 13. Model selection for a set of Gmail mouse click inter-events timing

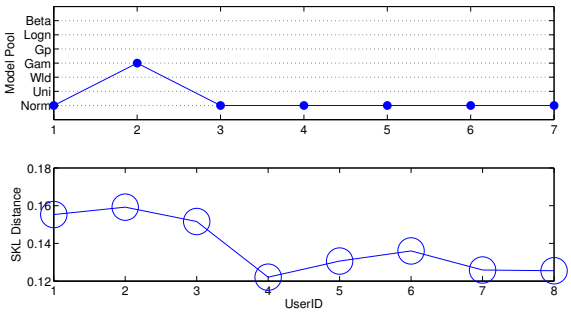

Figure 14. Model selection for a set of Gmail mouse angle

[5] N. Daswani and M. Stoppelman, "The anatomy of clickbot.A." in *Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets*, 2007.

[6] S. Kullback, "The Kullback-Leibler distance," *The American Statistician*, vol. 41, no. 4, pp. 340–341, 1987.

[7] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of ACM International Conference on Measurement and Modeling of Computer Science (SIGMETRICS)*, 1998, pp. 151–160.

[8] F. Yu, Y. Xie, and Q. Ke, "Sbotminer: Large scale search bot detection," in *Proceedings of ACM International Conference on Web Search and Data Mining*, 2010.

[9] P. M. Fitts and J. R. Peterson, "Information capacity of discrete motor responses," *Journal of Experimental Psychology*, vol. 67, no. 2, pp. 103–112, 1964.

[10] I. S. MacKenzie, "Fitts' law as a research and design tool in human-computer interaction," *Human-Computer Interaction*, vol. 7, pp. 91–139, 1992.

[11] U. Kukreja, W. E. Stevenson, and F. E. Ritter, "RUI: Recording user input from interfaces under Windows and Mac OS X," *Behavior Research Methods*, vol. 38, no. 4, pp. 656–659, 2006.

[12] J. Offutt and Y. Wu, "Modeling presentation layers of web applications for testing," *Software and Systems Modeling*, vol. 9, no. 2, pp. 257–280, April 2010.

[13] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge, UK: Cambridge University Press, 2008, ISBN 0-52188-038-1.

[14] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A symbolic execution framework for Javascript," in *Proceedings of the 31th IEEE Symposium on Security and Privacy*, 2010.

[15] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 2003, pp. 294–311.

[16] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, "reCAPTCHA: Human-Based Character Recognition via Web Security Measures," *Science*, vol. 321, no. 12, pp. 1465–1468, 2008.

[17] M. Motoyama, K. Levchenko, C. Kanich, D. Mccoy, G. M. Voelker, and S. Savage, "Re: Captchas – understanding captcha-solving services in an economic context," in *Proceedings of the 19th USENIX conference on Security*, 2010.

[18] Z. Chu, S. Gianvecchio, and H. Wang, "Who is tweeting on twitter: Human, bot, or cyborg?" in *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010.

[19] Z. Chu, S. Gianvecchio, A. Koehl, H. Wang, and S. Jajodia, "Blog or block: Detecting blog bots through behavioral biometrics," *Computer Networks*, vol. 57, no. 3, pp. 634–646, February 2013.

[20] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time URL spam filtering service," in *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011, pp. 447–462.

[21] H. Gao, Y. Chen, K. Lee, D. Palsetia, and A. Choudhary, "Towards online spam filtering in social networks," in *Proceedings of 19th Network and Distributed System Security Symposium (NDSS)*, 2012.

[22] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna, "Pubcrawl: protecting users and businesses from crawlers," in *Proceedings of the 21st USENIX Symposium on Security*, 2012.

[23] J. Jin and X. Wang, "On the effectiveness of low latency anonymous networks in the presence of timing attacks," in *Proceedings of the 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, July 2009, pp. 429–438.

[24] L. Ballard, F. Monrose, and D. Lopresti, "Biometric authentication revisited: understanding the impact of wolves in sheep's clothing," in *Proceedings of the 15th USENIX Symposium on Security*, 2006.

[25] M. Lee, R. R. Kompella, and S. Singh, "Ajaxtracker: active measurement system for high-fidelity characterization of Ajax applications," in *Proceedings of the 2010 USENIX conference on Web application development (WebApps'10)*, 2010.