Click Fraud Detection on the Advertiser Side

Haitao Xu[1], Daiping Liu[1], Aaron Koehl[1],
Haining Wang[1], and Angelos Stavrou[2]

[1] College of William and Mary, Williamsburg, VA 23187, USA
{hxu,liudptl,amkoeh,hnw}@cs.wm.edu
[2] George Mason University, Fairfax, VA 22030, USA
astavrou@gmu.edu

Abstract. Click fraud—malicious clicks at the expense of pay-per-click advertisers—is posing a serious threat to the Internet economy. Although click fraud has attracted much attention from the security community, as the direct victims of click fraud, advertisers still lack effective defense to detect click fraud independently. In this paper, we propose a novel approach for advertisers to detect click frauds and evaluate the return on investment (ROI) of their ad campaigns without the helps from ad networks or publishers. Our key idea is to proactively test if visiting clients are full-fledged modern browsers and passively scrutinize user engagement. In particular, we introduce a new functionality test and develop an extensive characterization of user engagement. Our detection can significantly raise the bar for committing click fraud and is transparent to users. Moreover, our approach requires little effort to be deployed at the advertiser side. To validate the effectiveness of our approach, we implement a prototype and deploy it on a large production website; and then we run 10-day ad campaigns for the website on a major ad network. The experimental results show that our proposed defense is effective in identifying both clickbots and human clickers, while incurring negligible overhead at both the server and client sides.

Keywords: Click Fraud, Online Advertising, Feature Detection.

1 Introduction

In an online advertising market, advertisers pay ad networks for each click on their ads, and ad networks in turn pay publishers a share of the revenue. As online advertising has evolved into a multi-billion dollar business [1], click fraud has become a serious and pervasive problem. For example, the botnet "Chameleon" infected over 120,000 host machines in the U.S. and siphoned $6 million per month from advertisers [2].

Click fraud occurs when miscreants make HTTP requests for destination URLs found in deployed ads [3]. Such HTTP requests issued with malicious intent are called fraudulent clicks. The incentive for fraudsters is to increase their own profits at the expense of other parties. Typically a fraudster is a publisher or an advertiser. Publishers may put excessive ad banners on their pages

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part II, LNCS 8713, pp. 419–438, 2014.

and then forge clicks on ads to receive more revenue. Unscrupulous advertisers make extensive clicks on a competitor's ads with the intention of depleting the victim's advertising budget. Click fraud is mainly conducted by leveraging clickbots, hiring human clickers, or tricking users into clicking ads [4].

In an act of click fraud, both an ad network and a publisher are beneficiaries while an advertiser is the only victim, under the pay-per-click model. Although the ad network pays out to the publisher for those undetected click fraud activities, it charges the advertiser more fees. Thus, the ad network still benefits from click fraud. Only the advertiser is victimized by paying for those fraudulent clicks. Therefore, advertisers have the strongest incentive to counteract click fraud. In this paper, we focus on click fraud detection from the perspective of advertisers.

Click fraud detection is not trivial. Click fraud schemes have been continuously evolving in recent years [3–7]. Existing detection solutions attempt to identify click fraud activities from different perspectives, but each has its own limitations. The solutions proposed in [8–10] perform traffic analysis on an ad network's traffic logs to detect publisher inflation fraud. However, an advanced clickbot can conduct a low-noise attack, which makes those abnormal-behavior-based detection mechanisms less effective. Haddadi [11] proposed exploiting bait ads to blacklist malicious publishers based on a predefined threshold. Motivated by [11], Dave et al. [4] proposed an approach for advertisers to measure click-spam ratios on their ads by creating bait ads. However, running bait ads increases advertisers' budget on advertisements.

In this paper, we propose a novel approach for an advertiser to independently detect click fraud attacks conducted by clickbots and human clickers. Our approach enables advertisers to evaluate the return on investment (ROI) of their ad campaigns by classifying each incoming click traffic as fraudulent, casual, or valid. The rationale behind our design lies in two observed invariants of legitimate clicks. The first invariant is that a legitimate click should be initiated by a real human user on a real browser. That is, a client should be a real full-fledged browser rather than a bot, and hence it should support JavaScript, DOM, CSS, and other web standards that are widely followed by modern browsers. The second invariant is that a legitimate ad clicker interested in advertised products must have some level of user engagement in browsing the advertised website.

Based on the design principles above, we develop a click fraud detection system mainly composed of two components: (1) a proactive functionality test and (2) a passive examination of browsing behavior. The functionality test actually challenges a client for its authenticity (a browser or a bot) with the assumption that most clickbots have limited functionality compared to modern browsers and thus would fail this test. Specifically, a client's functionality is validated against web standards widely supported by modern browsers. Failing the test would induce all clicks generated by the client to be labelled as fraudulent. The second component passively examines each user's browsing behaviors on the advertised website. Its objective is to identify human clickers and those more advanced clickbots that may pass the functionality test. If a client passes the functionality test and also shows enough browsing engagement on the advertised website, the corresponding click is labelled as valid. Otherwise, a click is labelled as casual if the corresponding client passes the functionality test but shows insufficient browsing behaviors. A casual click could be generated by a human clicker or by an unintentional user. We have no attempt to distinguish these two since neither of them is a potential customer from the standpoint of advertisers.

To evaluate the effectiveness of the proposed detection system, we build a prototype and deploy it on a large production web server. Then we run ad campaigns at one major ad network for 10 days. The experimental results show that our approach can detect much more fraudulent clicks than the ad network's in-house detection system and achieve low false positive and negative rates. We also measure the performance overhead of our detection system on the client and server sides.

Note that our detection mechanism can significantly raise the bar for committing click fraud and is potentially effective in the long run after public disclosure. To evade our detection mechanism, clickbots must implement all the main web standards widely supported by modern browsers. And a heavy-weight clickbot will risk itself of being readily noticeable by its host. Likewise, human clickers must behave like real interested users by spending more time, browsing more pages, and clicking more links on the advertised sites, which contradicts their original intentions of earning more money by clicking on ads as quickly as possible. At each point, the net effect is a disincentive to commit click fraud.

The remainder of the paper is organized as follows. We provide background knowledge in Section 2. Then, we detail our approach in Section 3 and validate its efficacy using real-world data in Section 4. We discuss the limitations of our work in Section 5 and survey related work in Section 6. Finally, we conclude the paper in Section 7.

2 Background

Based on our understanding of the current state of the art in click fraud, we first characterize clickbots and human clickers, the two main actors leveraged to commit click fraud. We then discuss the advertiser's role in inhibiting click fraud. Finally, we describe the web standards widely supported by modern browsers, as well as feature detection techniques.

2.1 Clickbots

A clickbot behaves like a browser but usually has relatively limited functionality compared to the latter. For instance, a clickbot may not be able to parse all elements of HTML web pages or execute JavaScript and CSS scripts. Thus, at the present time, a clickbot is instantiated as malware implanted in a victim's computer. Even assuming a sophisticated clickbot equipped with capabilities close to a real browser, its actual browsing behavior when connected to the advertised website would still be different from that of a real user. This is because

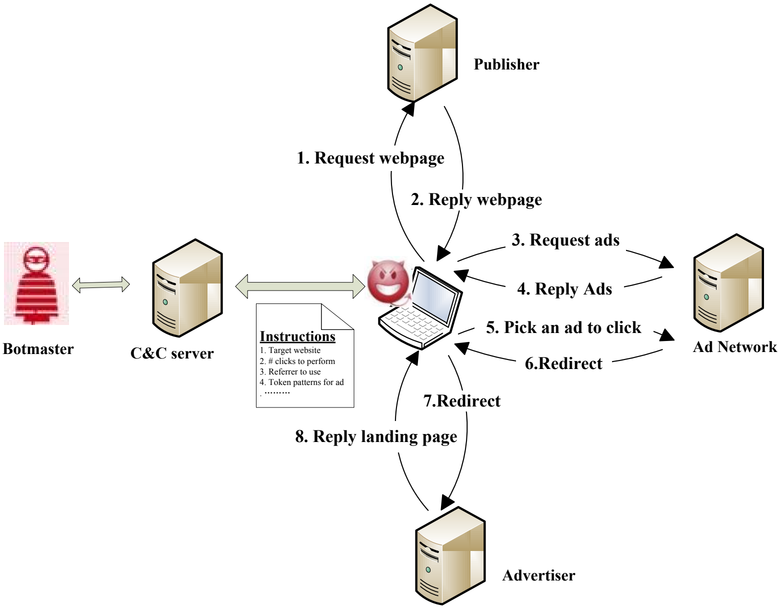

Fig. 1. How a clickbot works

clickbots are automated programs and are not sophisticated enough to see and think as human users, and as of yet, do not behave as human users.

A typical clickbot performs some common functions including initiating HTTP requests to a web server, following redirections, and retrieving contents from a web server. However, it does not have the ability to commit click fraud itself but instead acts as a relay based on instructions from a remote bot master to complete click fraud. A bot master can orchestrate millions of clickbots to perform automatic and large-scale click fraud attacks.

Figure 1 illustrates how a victim host conducts click fraud under the command of a botmaster. First, the botmaster distributes malware to the victim host by exploiting the host's security vulnerabilities, by luring the victim into a drive-by download or running a Trojan horse program. Once compromised, the victim host becomes a bot and receives instructions from a command-and-control (C&C) server controlled by the botmaster. Such instructions may specify the target website, the number of clicks to perform on the website, the referrer to be used in the fabricated HTTP requests, what kind of ads to click on, and when or how often to click [3].

After receiving instructions, the clickbot begins traversing the designated publisher website. It issues an HTTP request to the website (**step 1**). The website returns the requested page as well as all embedded ad tags on the page (**step 2**). An ad tag is a snippet of HTML or JavaScript code representing an ad, usually in an iframe. For each ad tag, the clickbot generates an HTTP request to the ad network to retrieve ad contents just like a real browser (**step 3**). The ad network returns ads to the clickbot (**step 4**). From all of the returned ads, the clickbot selects an ad matching the specified search pattern and simulates a click on the ad, which triggers another HTTP request to the ad network (**step 5**). The ad network logs the click traffic for the purpose of billing the advertiser and paying the publisher a share, and then returns an HTTP 302 redirect response (**step 6**). The clickbot follows the redirection path (possibly involving multiple parties) and finally loads the advertised website (**step 7**). The advertiser returns back the landing page[1] to the clickbot (**step 8**). At this point, the clickbot completes a single act of click fraud. Every time an ad is "clicked" by a clickbot, the advertiser pays the ad network and the involved publisher receives remuneration from the ad network. Note that a clickbot often works in the background to avoid raising suspicion, thus all HTTP requests in Figure 1 are generated without the victim's awareness.

2.2 Human Clickers

Human clickers are the people who are hired to click on the designated ads and get paid in return. Human clickers have financial incentives to click on ads as quickly as possible, which distinguishes them from real users who are truly interested in the advertised products. For instance, a real user tends to read, consider, think, and surf the website in order to learn more about a product before purchase. A paid clicker has few such interests, and hence tends to get bored quickly and spends little time on the site [12].

2.3 Advertisers

Advertisers are in a vantage point to observe and further detect all fraudulent activities committed by clickbots and human clickers. To complete click fraud, all fraudulent HTTP requests must be finally redirected to the advertised website, no matter how many intermediate redirections and parties are involved along the way. This fact indicates that both clickbots and human clickers must finally communicate with the victim advertiser. Thus, advertisers have the advantage of detecting clickbots and human clickers in the course of communication. In addition, as the revenue source of online advertising, advertisers have the strongest motivation to counteract click fraud.

2.4 Web Standards and Feature Detection Techniques

The main functionality of a browser is to retrieve remote resources (HTML, style, and media) from web servers and present those resources back to a user [13]. To correctly parse and render the retrieved HTML document, a browser should be compliant with HTML, CSS, DOM, and JavaScript standards which are represented by scriptable objects. Each object is attached with features including properties, methods, and events. For instance, the features attached to the DOM object include createAttribute, getElementsByTagName, title, domain, url, and

[1] Landing page is a single web page that appears in response to clicking on an ad.

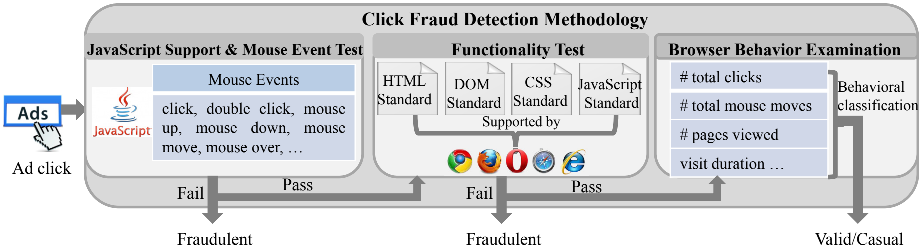

Fig. 2. Outline of click fraud detection mechanism

many others. Every modern browser supports those features. However, different browser vendors (and different versions) vary in support levels for those web standards, or they implement proprietary extensions all their own. To ensure that websites are displayed properly in all mainstream browsers, web developers usually use a common technique called feature detection to help produce JavaScript code with cross-browser compatibility.

Feature detection is a technique that identifies whether a feature or capability is supported by a browser's particular environment. One of the common techniques used is reflection. If the browser does not support a particular feature, JavaScript engines return null when referencing the feature; otherwise, JavaScript returns a non-null string. For instance, if the JavaScript statement "document.createElement" returns null in a specific browser, it indicates that the browser does not support the method createElement attached to the document object. Likewise, by testing a browser against a large number of fundamental features specified in web standards for modern browsers, we can estimate the browser's support level for those web standards, which helps validate the authenticity of the execution environment as a real browser.

Feature detection techniques have three primary advantages. First, feature detection can be an effective mechanism to detect clickbots. A clickbot cannot "pass" the feature detection unless it has implemented the main functionality of a real browser. Second, feature detection stresses the client's functionality thoroughly, and even a large pool of features can be used for feature detection in a fast and efficient manner. Lastly, the methods used for feature detection are designed to work across different browsers and will continue to work over time as new browsers appear, because new browsers fundamentally support reflection—even before implementing other features—and should also extend, rather than replace, existing web standards.

3 Methodology

Our approach mainly challenges a visiting client and its user engagement on the advertised site to determine whether the corresponding ad click is valid or not. To maximize detection accuracy, we also check the legitimacy of the origin (client's IP address) and the intermediate path (i.e., the publisher) of a click.

Figure 2 provides an outline of our approach. Our detection system consists of three components: (1) JavaScript support and mouse event test, (2) browser functionality test, and (3) browsing behavior examination.

For each incoming user, on the landing page, we test if the client supports JavaScript and if any mouse events are triggered. No JavaScript support or no mouse event indicates that the client may not be a real browser but a clickbot. Otherwise, we further challenge the client's functionality against the web standards widely supported by mainstream browsers. The client failed the functionality test is labelled as a clickbot. Otherwise, we further examine the client's browsing behavior on the advertiser's website and train a behavior-based classifier to distinguish a really interested user from a casual one.

3.1 JavaScript Support and Mouse Event Test

One simple way to detect clickbots is to test whether a client supports JavaScript or not. This is due to the fact that at least 98% of web browsers have JavaScript enabled [14] and online advertising services usually count on JavaScript support.

Monitoring mouse events is another effective way to detect clickbots. In general, a human user with a non-mobile platform (laptop/desktop) must generate at least one mouse event when browsing a website. A lack of mouse events flags the visiting client as a clickbot. However, this may not be true for users from mobile platforms (smartphones/pads). Thus, we only apply the mouse event test to users from non-mobile platforms.

Table 1. Tested browsers, versions and release dates

Chrome(10)	1.0.154 4/24/2009	2.0.173 6/23/2009	4.0.223 10/24/2009	5.0.307.1 1/30/2010	8.0.552.215 12/2/2010
	12.0.742.100 6/14/2011	16.0.912.63 12/7/2011	20.0.1132.47 6/28/2012	24.0.1312.57 1/30/2013	27.0.1453.94 5/24/2013
Firefox(10)	2.0 10/24/2006	3.0 6/17/2008	3.5 6/30/2009	3.6 1/21/2010	4.0 3/22/2011
	7.0 9/27/2011	11.0 3/13/2012	15.0 8/28/2012	19.0.2 3/7/2013	20.0.1 4/11/2013
IE(5)	6.0 8/27/2001	7.0 10/18/2006	8.0 3/19/2009	9.0 3/14/2011	10.0 10/26/2012
Safari(10)	3.1 3/18/2008	3.2 11/14/2008	3.2.2 2/15/2009	4.0 6/18/2009	4.0.5 3/11/2010
	5.0.1 7/28/2010	5.0.3 11/18/2010	5.1 7/20/2011	5.1.2 11/30/2011	5.1.7 5/9/2012
Opera(10)	8.50 9/20/2005	9.10 12/18/2006	9.20 4/11/2007	9.50 6/12/2008	10.00 9/1/2009
	10.50 3/2/2010	11.00 12/16/2010	11.50 6/28/2011	12.00 6/14/2012	12.15 4/4/2013

3.2 Functionality Test

A client passing the JavaScript and mouse event test is required to further undergo a feature-detection based functionality test.

Table 2. Authentic feature set widely supported by modern browsers

Objects	Features
Browser Window (51)	closed, defaultStatus, document, frames, history, alert, blur, clearInterval, clearTimeout, close, confirm, focus, moveBy, moveTo, open, print, prompt, resizeBy, resizeTo, scroll, scrollBy, scrollTo, setInterval, setTimeout, appCodeName, appName, appVersion, cookieEnabled, platform, userAgent, javaEnabled, availHeight, vailWidth, colorDepth, height, width, length, back, forward, go, hash, host, hostname, href, pathname, port, protocol, search, assign, reload, replace
DOM(26)	doctype, implementation, documentElement, createElement, createDocumentFragment, createTextNode, createComment, createAttribute, getElementsByTagName, title, referrer, domain, URL, body, images, applets, links, forms, anchors, cookie, open, close, write, writeln, getElementById, getElementsByName
CSS(76)	backgroundAttachment, backgroundColor, backgroundImage, backgroundRepeat, border, borderStyle, borderTop, borderRight, borderBottom, borderLeft, borderTopWidth, borderRightWidth, borderBottomWidth, borderLeftWidth, borderWidth, clear, color, display, font, fontFamily, fontSize, fontStyle, fontVariant, fontWeight, height, letterSpacing, lineHeight, listStyle, listStyleImage, listStylePosition, listStyleType, margin, marginTop, marginRight, marginBottom, marginLeft, padding, paddingTop, paddingRight, paddingBottom, paddingLeft, textAlign, textDecoration, textIndent, textTransform, verticalAlign, whiteSpace, width, wordSpacing, backgroundPosition, borderCollapse, borderTopColor, borderRightColor, borderBottomColor, borderLeftColor, borderTopStyle, borderRightStyle, borderBottomStyle, borderLeftStyle, bottom, clear, clip, cursor, direction, left, minHeight, overflow, pageBreakAfter, pageBreakBefore, position, right, tableLayout, top, unicodeBidi, visibility, zIndex

To avoid false positives and ensure that each modern browser can pass the functionality test, we perform an extensive feature support measurement on the top 5 mainstream browsers [15]: Chrome, Firefox, IE, Safari, and Opera. To discern the consistently supported features, we uniformly select 10 versions for each browser vendor with the exception of 5 versions for IE. Table 1 lists the browsers we tested. As a result, we obtain a set of 153 features associated with web standards, including browser window, DOM, and CSS (see Table 2). All those features are supported by both desktop browsers and their mobile versions. These features are commonly and consistently supported by the 45 versions of browsers in the past ten years. We call this set the authentic-feature set. We also create a bogus-feature set, which has the same size as the authentic-feature set but is obtained by appending "123" to each feature in the authentic-feature set. Thus, every feature in the bogus-feature set should not be supported by any real browser. Note that we just use the string "123" as an example. When implementing our detection, the advertiser should periodically change the string to make the bogus-feature set hard to evade.

How to Perform the Functionality Test. Figure 3 illustrates how the functionality test is performed. For the first HTTP request issued by a client, the advertiser's web server challenges the client by responding as usual, but along with a mixed set of authentic and bogus features. While the size of the mixed set is fixed (e.g.,100), the proportion of authentic features in the set is randomly decided. Then, those individual authentic and forged features in the set are randomly selected from the authentic and bogus feature sets, respectively. The client is expected to test each feature in its environment and then report to the web server how many authentic features are in the mixed set as the response to the challenge.

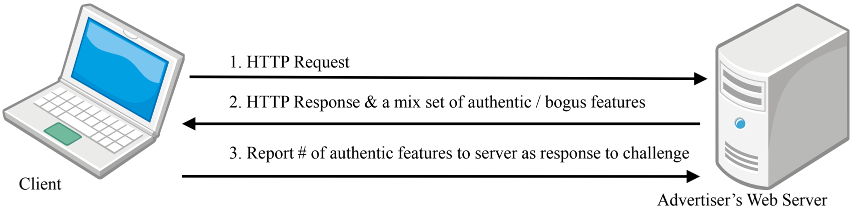
1. HTTP Request
2. HTTP Response & a mix set of authentic / bogus features
3. Report # of authentic features to server as response to challenge
Client
Advertiser's Web Server

Fig. 3. How the functionality test is performed by advertiser's web server

A real browser should be able to report the correct number of authentic features to the web server after executing the challenge code, and thus passes the functionality test. However, a clickbot would fail the test because it is unable to test the features contained in the set and return the correct number. Considering some untested browsers may not support some authentic features, we set up a narrow range $[x - N, x]$ to handle this, where x is the expected number and N is a small non-negative integer. A client is believed to pass the test as long as its reported number falls within $[x - N, x]$. Here we set N to 4 based on our measurement results.

Evasion Analysis. Assume that a client receives a mixed set of 150 features from a web server and the set consists of 29 randomly selected authentic features and 121 randomly selected bogus features. Thus, the expected number should fall into the range [25,29]. Consider a crafty clickbot who knows about our detection mechanism in advance. The clickbot does not need to test the features, but just guesses a number from the possible range [0,150], and returns it to the server. In this case, the probability for the guessed number to successfully fall into [25,29] is only 3%. Thus, the clickbot has little chance (3%) to bypass the functionality test.

3.3 Browsing Behavior Examination

Passing the functionality test cannot guarantee that a click is valid. An advanced clickbot may function like a real browser and thus can circumvent the functionality test. A human clicker with a real browser can also pass the test.

However, clickbots and human clickers usually show quite different browsing behaviors on the advertised website from those of real users. Click fraud activities conducted by clickbots usually end up with loading the advertiser's landing page and do not show human behaviors on the site. For human clickers, their only purpose is to make more money by clicking on ads as quickly as possible. They tend to browse an advertised site quickly and then navigate away for the next click task. Instead, real interested users tend to learn more about a product and spend more time on the advertised site. They usually scroll up and down a page, click on their interested links, browses multiple pages, and sometimes make a purchase.

Therefore, we leverage users' browsing behaviors on the advertised site to detect human clickers and advanced clickbots. Specifically, we extract extensive

Table 3. Summary of our ad campaigns

Set	Campaign	Clicks	Impressions	CTR	Invalid Clicks	Invalid Rate	Avg. CPC	Daily Budget	Duration (days)
1	bait1	1,011	417,644	0.24%	425	29.60%	$0.08	$15.00	10
2	bait2	4,127	646,152	0.64%	852	17.11%	$0.03	$15.00	10
3	bait3	5,324	933,790	0.57%	1,455	21.46%	$0.04	$15.00	10
4	normal1	288	68,425	0.42%	18	5.88%	$0.40	$20.00	10
5	normal2	224	20,784	1.08%	10	4.27%	$0.48	$20.00	10
Total	NA	10,974	2,086,795	0.53%	2,760	25.15%	$0.06	$85.00	10

features from passively collected browsing traffic on the advertised website, and train a classifier for detection.

4 Experimental Results

In order to evaluate our approach, we run ad campaigns to collect real-world click traffic, and then analyze the collected data to discern its primary characteristics, resulting in a technique to classify click traffic as either fraudulent, casual, or valid.

4.1 Running Ad Campaigns

To obtain real-world click traffic, we signed up with a major ad network and ran ad campaigns for a high-traffic woodworking forum website. Motivated by the bait ad technique proposed in [11], we created three bait ads for the site and made the same assumption as the previous works [4, 11, 16], that very few people would intentionally click on the bait ads and those ads are generally clicked by clickbots and fraudulent human clickers. Bait ads are textual ads with nonsense content, as illustrated in Figure 4. Note that our bait ads were generated in English. In addition, we created two normal ads, for which the ad texts describe the advertised site exactly. Our goal of running ad campaigns is to acquire both malicious and authentic click traffic for validating our click fraud detection system. To this end, we set the bait ads to be displayed on partner websites of any language across the world but display normal ads only on search result pages in English to avoid publisher fraud cases from biasing the clicks on the latter normal ads. We expect that most, if not all, clicks on bait ads and normal ads are fraudulent and authentic, respectively.

We ran our ad campaigns for 10 days. Table 3 provides a summary of our ad campaigns. Our ads had 2 million impressions[2], received nearly 11 thousand clicks and had a click-through rate (CTR) of 0.53% on average. Among these, 2.7 thousand clicks were considered by the ad network as illegitimate and were not charged. The invalid click rate was 25.15%. The average cost per click (CPC) was $0.06. Note that the two normal ads only received 512 clicks accounting for 4.67% of the total. The reason is that although we provided quite high bids for

[2] An ad being displayed once is counted as one impression.

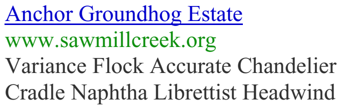
Anchor Groundhog Estate
www.sawmillcreek.org
Variance Flock Accurate Chandelier
Cradle Naphtha Librettist Headwind

Fig. 4. A bait ad with the ad text of randomly selected English words

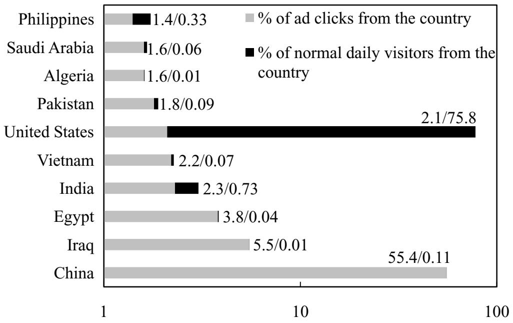

Fig. 5. Distribution of click traffic vs. that of normal traffic by country

normal ads, our normal ads still cannot compete with those of other advertisers for top positions and thus received fewer clicks.

4.2 Characterizing the Click Traffic

We characterize the received click traffic by analyzing users' geographic distribution, browser type, IP address reputation, and referrer websites' reputations. Our goal, through statistical analysis, is to have a better understanding of both the users who clicked on our ads and the referrer websites where our ads were clicked. Although the ad network reported that our ads attracted close to 11 thousand clicks, we only caught on the advertised site 9.9 thousand clicks, which serve as data objects for both closer examination and validation of our approach.

Geographic Distribution. We obtain users' geographic information using an IP geolocation lookup service [17]. Our 9.9 thousand clicks originate from 156 countries. Figure 5 shows the distribution of ad clicks by the top 10 countries which generate the most clicks. The distribution of normal daily visitors to the advertised site by country is also given in Figure 5. Note that the data form 'X/Y' means that $X\%$ of ad clicks and $Y\%$ of normal daily visitors are from that specific country. The top 10 countries contribute 77.7% of overall clicks. China alone contributes over 55% of the clicks, while the United States contributes 2.1%. This is quite unusual because the normal daily visitors from China only account for 0.11% while the normal visitors from the United States close to 76%. Like China, Egypt, Iraq, and other generally non-English countries also contribute much higher shares of ad click traffic than their normal daily traffic to the site. The publisher websites from these countries are suspected to be using bots to click on our ads. Even worse, one strategy of our ad network partner may aggravate the fraudulent activities. The strategy says that when an ad has a high click through ratio on a publisher website, the ad network will deliver the ad to that publisher website more frequently. To guarantee that our ads attract as

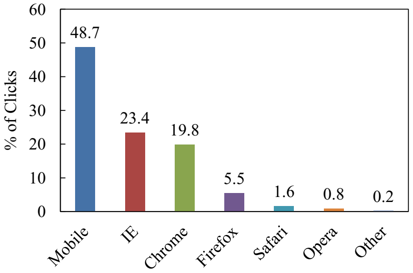

Fig. 6. Distribution of click traffic by browser

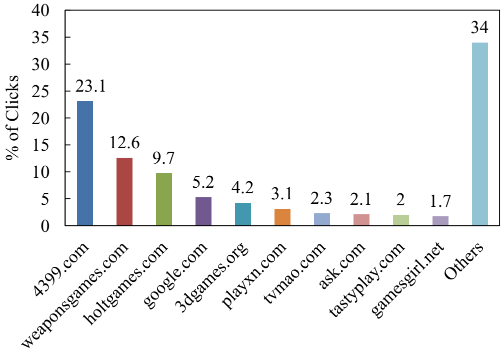

Fig. 7. Distribution of click traffic by publisher

many clicks as possible within a daily budget, the ad network may deliver our ads to those non-English websites more often.

Browser Type. Next we examine the distribution of the browsers to see which browser vendors are mostly used by users to view and click on our ads. We extracted the browser information from the User-Agent strings of the HTTP requests to our advertised website.

Figure 6 shows the distribution of the browsers used by our ad clickers. IE, Chrome, Firefox, Safari, and Opera are the top 5 desktop and laptop browsers, which is consistent with the web browser popularity statistics from StatCounter [15]. Notably, mobile browsers alone contribute to nearly 50% of overall traffic, much larger than the estimated usage share of mobile browsers (about 18% [18]). Close scrutinization reveals that 40% of the traffic with mobile browsers originates from China. China generated over 50 percent of overall traffic, which skews the browser distribution.

Blacklists. A fraction of our data could be generated by clickbots and compromised hosts. Those malicious clients could also be utilized by fraudsters to conduct other undesirable activities, and are thus blacklisted. By looking up users' IP addresses in public IP blacklists [19], we found that 29% of the total hosts have ever been blacklisted.

Referrers. Another interesting question would be which websites host our ads and if their contents are really related to the keywords of our ads. According to the contextual targeting policy of the ad network, an ad should be delivered to the ad network's partner websites whose contents match the selected keywords for the ad.

We used the Referer field in the HTTP request header to locate the publishers that displayed our ads and then directed users to our advertised website. However, we can only identify publishers for only 37.2% of the traffic (3,685 clicks) because the remaining traffic either has a blank Referer field or has the domain of the ad network as the referer field. For example, the Referer field for more than 40% of traffic has the form of doubleclick.net. We then examined, among those detected publishers, which websites contribute to the most clicks. Note that publishers could be websites or mobile apps. We identified 499 unique websites and 5 apps in total. Those apps are all iPhone apps and only generate 28 clicks all together. The remaining 3,657 clicks are from the 499 unique websites. Figure 7 shows the distribution of the click traffic by those 504 publishers. The top 3 websites with the most clicks on our ads are all small game websites, which contribute to over 45% of publisher-detectable clicks. Actually, the top 7 websites are all small game websites. Small game websites often attract many visitors, and thus the ads on those websites are more likely to be clicked on. However, our keywords are all woodworking-related and evidently, the contents of those game websites do not match our keywords. According to the above mentioned contextual targeting policy, the ad network should have not delivered our ads to such websites. One possible reason is that from the perspective of the ad network, attracting clicks takes precedence over matching the ads with host websites.

4.3 Validating Detection Approach

As described before, our approach is composed of three main components: a JavaScript support and mouse event test, a functionality test, and a browsing behavior examination. Here we individually validate their effectiveness.

JavaScript Support and Mouse Event Test. Among the 9.9 thousand ad clicks logged by the advertised site, 75.2% of users do not support JavaScript. We labelled those users as clickbots. Note that this percentage may be slightly overestimated considering that some users (at most 2% [14]) may have JavaScript disabled. In addition, those visits without support for JavaScript do not correlate with visits from mobile browsers. We have checked that nearly all mobile browsers provide support for JavaScript despite limited computing power. We then focused on the top 10 publisher websites with the most clicks to identify potentially malicious publishers. Figure 8 depicts the percentage of clicks without script support from those top 10 publishers. Among them, the two non-entertainment websites google.com and ask.com have low ratios, 9.4% and 15.2%, respectively. In contrast, the other 8 entertainment websites have quite high click ratios without script support. There are 86 visits from tvmao.com and none of them support JavaScript. We believe that all 86 clicks are fraudulent and generated by bots. Similarly, 99.1% of clicks from weaponsgames.com, 96.1% of clicks from 3dgames.org, and 95.3% from gamesgirl.net are without JavaScript support either. Such high ratios indicate that the invalid click rate in the real-world ad campaigns is much larger than the average invalid rate of 25.15% alleged by the ad network for our ad campaigns, as shown in Table 3.

We observed 506 ad clicks (with JavaScript support) that result in zero mouse events when arriving at our target site. Of those, 96 are initiated from mobile platforms including iPad, iPhone, Android, and Windows Phone. The remaining

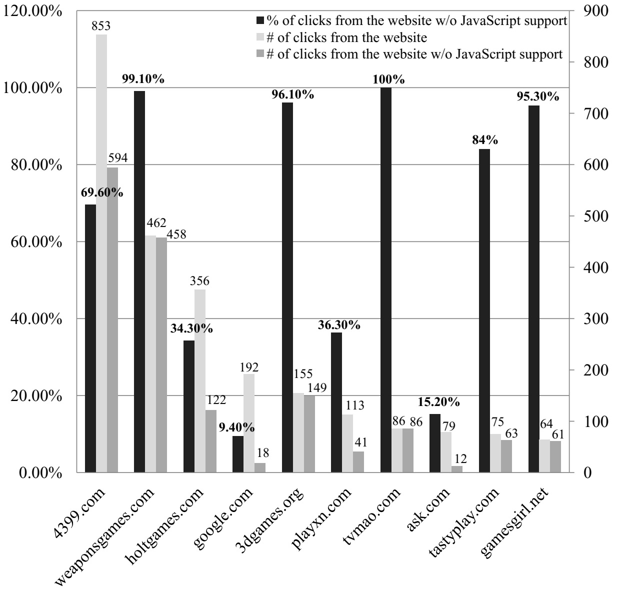

Fig. 8. Percentage of clicks without JavaScript support for the top 10 publisher websites contributing the most clicks

410 clicks are generated from desktop or laptop platforms. Those 410 ad clicks also have few other kinds of user engagement: no mouse clicks, no page scrolls, and short dwelling time. We labelled them as clickbots.

We further investigated the click traffic from 4399.com due to the fact that this website generated the most clicks on our ads among all identified publishers. The following several pieces of data indicate the existence of publisher fraud. First, all 853 clicks from 4399.com were generated within one day. Notably, up to 95 clicks were generated within one hour. Second, several IPs were found to click on our ads multiple times within one minute using the same User-Agent, and one User-Agent was linked to almost 15 clicks on average. Third, close to 70% of clients did not support JavaScript. Hence we suspect that the website owner used automated scripts to generate fraudulent clicks on our ads. However, the scripts are likely incapable of executing the JavaScript code attached to our ads. In addition, they probably spoofed IP address and User-Agent fields in the HTTP requests to avoid detection.

Functionality Test. The clickbots that cannot work as full-fledged modern browsers are expected to fail our functionality test. Among the logged 9.9 thousand clicks, 7,448 clicks without JavaScript support did not trigger the functionality test, and 35 of the remaining clicks with JavaScript support were observed to fail the functionality test and were subsequently labelled as clickbots. So far, 75.6% of clicks (7,483 clicks) had been identified by our detection mechanism to originate from clickbots. Among them, 99.5% (7,448 clicks) were simple click-

Table 4. Features extracted for each ad click

Feature Category	Feature Description
Mouse clicks	# of total clicks made on the advertised site # of clicks made only on the pages excluding the landing page # of clicks exclusively made on hyperlinks
Mouse scrolls	# of scroll events in total # of scroll events made on the pages excluding the landing page
Mouse moves	# of mousemove events in total # of mousemove events made only on the pages excluding the landing page
Pages views	# of pages viewed by a user
Visit duration	How long a user stays on the site
Execution efficiency	Client's execution time of JavaScript code for challenge
Legitimacy of origin	If the source IP is in any blacklist
Publisher's reputation	If the click originates from an disreputable website

bots without JavaScript support; and the rest 0.5% (35 clicks) were relatively advanced clickbots with JavaScript support yet failed the functionality test.

Browsing Behavior Examination. After completing the two steps above and discarding incomplete click data, 1,479 ad clicks (14.9 %) are left to be labelled. Among them, 1,127 ad clicks are on bait ads while the other 352 clicks are on normal ads. Here we further classify the click traffic into three categories—fraudulent, casual, and valid—based on user engagement, client IP, and publisher reputation information.

Features. We believe that three kinds of features are effective to differentiate advanced clickbots and human clickers from real users. (1) How users behave at the advertised site, i.e., users' browsing behavior information. (2) Who clicks on our ads, and a host with a bad IP is more likely to issue fraudulent clicks. (3) Where a user clicks on ads, and a click originating from a disreputable website tends to be fraudulent. Table 4 enumerates all the features we extracted from each ad click traffic to characterize users' browsing behaviors on the advertised site.

Ground truth. Previous works [4, 11, 16] all assume that very few people would intentionally click on bait ads and only clickbots and human clickers would click on such ads. That is, a click on a bait ad is thought to be fraudulent. However, this assumption is too absolute. Consider the following situation. A real user clicks on a bait ad unintentionally or just out of curiosity, without malicious intention. Then, the user happens to like the advertised products and begins browsing the advertised site. In this case, the ad click generated by this user should not be labelled as fraudulent. Thus, to minimize false positives, we partly accept the above common assumption, scrutinize those bait ad clicks which have shown rich human behaviors on the advertised site, and correct a-priori labels based on the following heuristics. Specifically, for a bait ad click, if the host IP address is not in any blacklist and the referrer website has a good reputation, this ad click is relabelled as valid when one of the following conditions holds: (1) 30 seconds of dwelling time, 15 mouse events, and 1 click; (2) 30 seconds of dwelling time, 10 mouse events, 1 scroll event, and 1 click; and (3) 30 seconds of dwelling time, 10 mouse events, and 2 page views. We believe the above conditions are strict enough to avoid mislabelling the ad clicks generated by bots and human clickers as valid clicks.

Note that our normal ads are only displayed on the search engine result pages with the expectation that most, if not all, clicks on normal ads are valid. The ad campaign report provided by the ad network in Table 3 confirms this, showing that the invalid click rate for normal ads is only 5.08% on average. Based on our design and the ad campaign report, we basically assume that the clicks on normal ads are valid. However, after further manually checking the normal ad clicks, we found that some of them do not demonstrate sufficient human behaviors, and these normal ad clicks will be relabelled as casual when one of the following two conditions holds: (1) less than 5 seconds of dwelling time; (2) less than 10 seconds of dwelling time and less than 5 mouse events. The casual click traffic could be issued by human users who unintentionally click on ads and then immediately navigate away from the advertised site. From the advertisers' perspective, such a click traffic does not provide any value when evaluating the ROI of their ad campaigns on a specific ad network, and therefore should be classified as casual.

Actually, if there is no financial transaction involved, only a user's intention matters whether the corresponding ad click is fraudulent or not. That is, only users themselves know the exact ground truth for fraudulent/valid/casual clicks. For those clicks without triggering any financial transactions, we utilize the above reasonable assumptions and straightforward heuristics to form the ground truth for fraudulent/valid/casual clicks.

Evaluation metrics. We evaluated our detection against two metrics—false positive rate and false negative rate. A false positive is when a valid click is wrongly labelled as fraudulent, and a false negative is when a fraudulent click is incorrectly labelled as valid.

Classification results. Using Weka [20], we chose a C4.5 pruned decision tree [21] with default parameter values (i.e., 0.25 for confidence factor and 2 for minimum number of instances per leaf) as the classification algorithm, and ran a 10-fold cross-validation. The false positive rate and false negative rate were 6.1% and 5.6%, respectively. Note that these are the classification results on those 1,479 unlabelled clicks. As a whole, our approach showed a high detection accuracy on the total 9.9 thousand clicks, with a false positive rate of 0.79% and a false negative rate of 5.6%, and the overall detection accuracy is 99.1%.

Overhead. We assessed the overhead induced by our detection on the client and server sides, in terms of time delay, CPU, memory and storage usages.

The only extra work required of the client is the execution of a JavaScript challenge script and to report the functionality test results to the server as an AJAX POST request. We measured the overhead on the client side using two metrics: source lines of code (SLOC) and the execution time of JavaScript code. The JavaScript code is only about 150 SLOC and we observed negligible impact on the client. We also estimated the client's execution time of JavaScript from the server side to avoid the possibility that the client could report a bogus execution time. Note that the execution time measured by the server contains a round trip time, which makes the estimated execution time larger than the actual execution time. Figure 9 depicts the 9.9 thousand clients' execution time of the JavaScript challenge code. About 80% of clients finished execution within

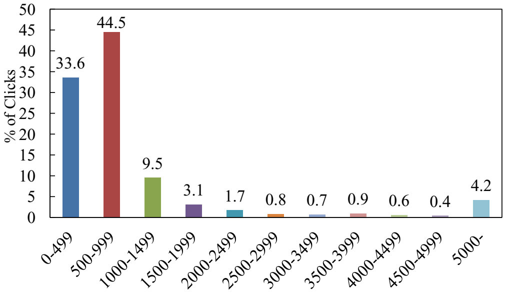

Fig. 9. Clients' execution time of JavaScript challenge code in milliseconds

one second. Assuming that the round trip time (RTT) is 200 milliseconds, the actual computation overhead incurred at the client side is merely several hundred milliseconds.

We used the SAR (System Activity Report) [22] to analyze server performance and measure the overhead on the server side. We observed no spike in server load. This is because most of work involved in our detection happens on the client side, and the induced click-related traffic is insignificant in comparison with server's normal traffic.

5 Discussion and Limitations

In this paper, we assume that a clickbot typically does not include its own JavaScript engine or access the full software stack of a legitimate web browser residing on the infected host. A sophisticated clickbot implementing a full browser agent itself would greatly increase its presence and the likelihood of being detected. A clickbot might also utilize a legitimate web browser to generate activities, and can thus pass our browser functionality test. To identify such clickbots, we could further detect whether our ads and the advertised websites are really visible to users by utilizing a new feature provided by some ad networks. The new feature allows advertisers to instrument their ads with JavaScript code for a better understanding of what is happening to their ads on the client side. With this feature, we could detect if our ad iframe is visible at the client's front-end screen rather than in the background, and if it is really focused and clicked on.

In addition, compared to our user-visit related features (dwelling time, mouse events, scroll events, clicks and etc.), user-conversation related features[3] are expected to have better discriminating power between clickbots, human clickers, and real users in browsing behaviors. However, our advertised site is a professional forum rather than an online retailer. If a user registers (creates an account) on the forum, it is analogous to a purchase at an online retailer. However, such conversion from guest to member is an event too rare to rely upon to enhance our classifier.

[3] Purchasing a product, abandoning an online cart, proactive online chat, etc.

6 Related Work

Browser Fingerprinting. Browser fingerprinting allows a website to identify a client browser even though the client disables cookies. Existing browser fingerprinting techniques could be mainly classified into two categories, based on the information they need for fingerprinting. The first category fingerprints a browser by collecting application-layer information, including HTTP request header information and system configuration information from the browser [23]. The second category performs browser fingerprinting by examining coarse traffic generated by the browsers [24]. However, both of them have their limitations in detecting clickbots. Nearly all the application-layer information can be spoofed by sophisticated clickbots, and browser fingerprints may change quite rapidly over time [23]. In addition, an advertiser often cannot collect enough traffic information for fingerprinting the client from just one visit to the advertiser. Compared to the existing browser fingerprinting techniques, our feature detection technique has three main advantages. First, clickbots cannot easily pass the functionality test unless they have implemented the main functionality present in modern browsers. Second, the client's functionality could be tested thoroughly at the advertiser's side even though the client visits the advertiser's landing page only once. Lastly, our technique works over time as new browsers appear because new browsers should also conform to the those web standards currently supported by modern browsers.

Revealed Click Fraud. Several previous studies investigate known click fraud activities, and clickbots have been found to be continuously evolving and become more sophisticated. As the first study to analyze the functionality of a clickbot, Daswani et al. [3] dissected Clickbot.A and found that the clickbot could carry out a low-noise click fraud attack to avoid detection. Miller et al. [5] examined two other families of clickbots. They found that these two clickbots were more advanced than Clickbot.A in evading click fraud detection. One clickbot introduces indirection between bots and ad networks, while the other simulates human web browsing behaviors. Some other characteristics of clickbots are described in [4]. Clickbots generate fraudulent clicks periodically and only issue one fraudulent click in the background when a legitimate user clicks on a link, which makes fraudulent traffic hardly distinguishable from legitimate click traffic. Normal browsers may also be exploited to generate fraudulent click traffic. The traffic generated by a normal browser could be hijacked by currently visited malicious publishers and be further converted to fraudulent clicks [7]. Ghost click botnet [6] leverages DNS changer malware to convert a victim's local DNS resolver into a malicious one and then launches ad replacement and click hijacking attacks. Our detection can identify each of these clickbots by actively performing a functionality test and can detect all other kinds of click fraud by examining their browsing behavior traffic on the server side.

Click Fraud Detection. Metwally et al. conducted an analysis on ad networks' traffic logs to detect publishers' non-coalition hit inflation fraud [8], coalition fraud [9], and duplicate clicks [10]. The main limitation of these works lies in that ad networks' traffic logs are usually not available to advertisers. Haddadi in [11] and Dave et al. in [4] suggested that advertisers use bait ads to detect fraudulent clicks on their ads. While bait ads have been proven effective in detection, advertisers have to spend extra money on those bait ads. Dave et al. [16] presented an approach to detecting fraudulent clicks from an ad network's perspective rather than an advertiser's perspective. Li et al. [7] introduced the ad delivery path related features to detect malicious publishers and ad networks. However, monitoring and reconstructing the ad delivery path is time-consuming and difficult to detect click frauds in real time. Schulte et al. [25] detected client-side malware using so-called program interactive challenge (PIC) mechanism. However, an intermediate proxy has to be introduced to examine all HTTP traffic between a client and a server, which would inevitably incur significant delay. Like [4, 11], our defense works at the server side but does not cause any extra cost for advertisers. Our work is the first to detect clickbots by testing their functionalities against the specifications widely conformed to by modern browsers. Most clickbots can be detected at this step, because they have either no such functionalities or limited functionalities compared to modern browsers. For the advanced clickbots and human clickers, we scrutinize their browsing behaviors on the advertised site, extract effective features, and train a classifier to identify them.

7 Conclusion

In this paper, we have proposed a new approach for advertisers to independently detect click fraud activities issued by clickbots and human clickers. Our proposed detection system performs two main tasks of proactive functionality testing and passive browsing behavior examination. The purpose of the first task is to detect clickbots. It requires a client to actively prove its authenticity of a full-fledged browser by executing a piece of JavaScript code. For more sophisticated clickbots and human clickers, we fulfill the second task by observing what a user does on the advertised site. Moreover, we scrutinize who initiates the click and which publisher website leads the user to the advertiser's site, by checking the legitimacy of the clients' IP addresses (source) and the reputation of the referring site (intermediate), respectively. We have implemented a prototype and deployed it on a large production website for performance evaluation. We have then run a real ad campaign for the website on a major ad network, during which we characterized the real click traffic from the ad campaign and provided advertisers a better understanding of ad click traffic, in terms of geographical distribution and publisher website distribution. Using the real ad campaign data, we have demonstrated that our detection system is effective in the detection of click fraud.

References

1. https://en.wikipedia.org/wiki/Online_advertising
2. http://www.spider.io/blog/2013/03/chameleon-botnet/

3. Daswani, N., Stoppelman, M.: The anatomy of clickbot.a. In: Proceedings of the Workshop on Hot Topics in Understanding Botnets (2007)
4. Dave, V., Guha, S., Zhang, Y.: Measuring and fingerprinting click-spam in ad networks. In: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (2012)
5. Miller, B., Pearce, P., Grier, C., Kreibich, C., Paxson, V.: What's clicking what? techniques and innovations of today's clickbots. In: Holz, T., Bos, H. (eds.) DIMVA 2011. LNCS, vol. 6739, pp. 164–183. Springer, Heidelberg (2011)
6. Alrwais, S.A., Dun, C.W., Gupta, M., Gerber, A., Spatscheck, O., Osterweil, E.: Dissecting ghost clicks: Ad fraud via misdirected human clicks. In: Proceedings of the Annual Computer Security Applications Conference (2012)
7. Li, Z., Zhang, K., Xie, Y., Yu, F., Wang, X.: Knowing your enemy: Understanding and detecting malicious web advertising. In: Proceedings of the ACM Conference on Computer and Communications Security (2012)
8. Metwally, A.: Sleuth: Single-publisher attack detection using correlation hunting. In: Proceedings of the International Conference on Very Large Data Bases (2008)
9. Metwally, A.: Detectives: Detecting coalition hit inflation attacks in advertising networks streams. In: Proceedings of the International Conference on World Wide Web (2007)
10. Metwally, A., Agrawal, D., Abbadi, A.E.: Duplicate detection in click streams. In: Proceedings of the International Conference on World Wide Web (2005)
11. Haddadi, H.: Fighting online click-fraud using bluff ads. In: ACM SIGCOMM Computer Communication Review (2010)
12. Daswani, N., Mysen, C., Rao, V., Weis, S., Gharachorloo, K., Ghosemajumder, S.: Online advertising fraud. In: Crimeware: Understanding New Attacks and Defenses. Addison-Wesley Professional (2008)
13. http://taligarsiel.com/Projects/howbrowserswork1.htm
14. https://developer.yahoo.com/blogs/ydnfourblog/many-users-javascript-disabled-14121.html
15. http://gs.statcounter.com/
16. Dave, V., Guha, S., Zhang, Y.: Viceroi: Catching click-spam in search ad networks. In: Proceedings of ACM Conference on Computer and Communications Security (2013)
17. http://www.maxmind.com/en/web_services
18. http://en.wikipedia.org/wiki/Usage_share_of_web_browsers
19. http://www.blacklistalert.org/
20. http://www.cs.waikato.ac.nz/ml/weka/
21. Quinlan, J.: C4.5: Programs for machine learning. Morgan Kaufmann Publishers (1993)
22. http://en.wikipedia.org/wiki/Sar_Unix
23. Eckersley, P.: How unique is your web browser? In: Proceedings of the Privacy Enhancing Technologies Symposium (2010)
24. Yen, T.-F., Huang, X., Monrose, F., Reiter, M.K.: Browser fingerprinting from coarse traffic summaries: Techniques and implications. In: Flegel, U., Bruschi, D. (eds.) DIMVA 2009. LNCS, vol. 5587, pp. 157–175. Springer, Heidelberg (2009)
25. Schulte, B., Andrianakis, H., Sun, K., Stavrou, A.: Netgator: Malware detection using program interactive challenges. In: Flegel, U., Markatos, E., Robertson, W. (eds.) DIMVA 2012. LNCS, vol. 7591, pp. 164–183. Springer, Heidelberg (2013)