JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2016; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

Protecting internet users from becoming victimized attackers of click-fraud

Md Shahrear Iqbal[1]*, Mohammad Zulkernine[1], Fehmi Jaafar[2] and Yuan Gu[3]

[1]*School of Computing, Queen's University, Kingston, Ontario, Canada.*
[2] *Ubitrak, Saint-Laurent, Quebec, Canada.*
[3] *Irdeto, Ottawa, Ontario, Canada.*

ABSTRACT

Internet users are often victimized by malicious attackers. Some attackers infect and use innocent users' machines to launch large-scale attacks without the users' knowledge. One of such attacks is the click-fraud attack. Click-fraud happens in Pay-Per-Click (PPC) ad networks where the ad network charges advertisers for every click on their ads. Click-fraud has been proved to be a serious problem for the online advertisement industry. In a click-fraud attack, a user or an automated software clicks on an ad with a malicious intent and advertisers need to pay for those valueless clicks. Among many forms of click-fraud, botnets with the automated clickers are the most severe ones. In this paper, we present a method for detecting automated clickers from the user-side. The proposed method to **F**ight **C**lick-**Fraud**, **FCFraud**, can be integrated into the desktop and smart device operating systems. Since most modern operating systems already provide some kind of anti-malware service, our proposed method can be implemented as a part of the service. We believe that an effective protection at the operating system level can save billions of dollars of the advertisers. Experiments show that FCFraud is 99.6% (98.2% in mobile ad library generated traffic) accurate in classifying ad requests from all user processes and it is 100% successful in detecting clickbots in both desktop and mobile devices. We implement a cloud backend for the FCFraud service to save battery power in mobile devices. The overhead of executing FCFraud is also analyzed and we show that it is reasonable for both the platforms. Copyright © 2016 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: click-fraud; malware detection; online advertising

1. INTRODUCTION

Online advertising is a form of marketing which uses digital medium (e.g., websites and mobile apps) to deliver promotional messages to consumers. It includes email marketing, search engine marketing (SEM), social media marketing, and many types of display advertising. It is the main financial incentive for free web contents and services as well as free mobile apps. The online advertising industry is a billion dollar industry with ad spending projected to reach $161 billion in desktop and $101 billion in mobile devices by 2016 [1, 2]. The largest revenue shares within internet advertising are generated by display-based and search-based advertising.

Advertisers and web/mobile publishers use a wide range of revenue models. Among all the models, Pay-Per-Click (PPC) is the dominant one. In PPC, advertisers pay each time a user clicks on an ad. The biggest threat to the PPC advertisement is click-fraud [3]. Click-fraud

*Correspondence to: Md Shahrear Iqbal, Room: 536, School of Computing, Queen's University, Kingston, Ontario, Canada.
E-mail: iqbal@cs.queensu.ca

is the practice of deceptively clicking on online ads with the intention of either increasing website/mobile app revenues or exhausting an advertiser's budget. Click-fraud is a major concern for advertisers, and many researchers proposed methods to detect and block suspicious clicks on online advertisements [4, 5, 6, 7, 8].

Among all forms of click-fraud, attacks by the botnets are the most severe. Attackers use botnets extensively in recent years to launch large-scale attacks [9, 10, 11] like click-fraud. A botnet is a network of malware-infected devices that are controlled by a botmaster. The users are normally unaware of the fact that their devices are compromised and used by the attackers. FCFraud particularly protects this group of users from being exploited.

To combat click-fraud, ad networks mostly use server-side techniques. They gather information from different sources about users. Then, they apply machine learning or pattern recognition techniques to identify suspicious clicks. Despite all these efforts, click-fraud remains as a primary problem for the online advertising industry due to the lack of control over the client machines from the server-side. In fact, a large percentage of all web traffic is considered fake (non-human generated) and costs US advertisers 4.2 billion dollar annually [12].

This paper[†] proposes a method, FCFraud, which tries to solve the click-fraud problem from the user-side. FCFraud can be incorporated as a part of the operating system's anti-malware service. Although, desktop devices are used exclusively to launch large-scale click fraud attacks, we observe that smart devices (e.g., smartphones and smart TVs) are also capable of launching such attacks if compromised by malware. Click-fraud malware communicate with their botmasters using the internet and perform click-fraud stealthily. As there are billions of smart devices with wireless internet, the problem of click-fraud attack from mobile devices may have larger impact in the near future.

FCFraud inspects HTTP packets from all user processes along with the events from hardware input devices (mouse, keyboard, touchscreens, etc.). It analyzes the ad-related traffic from the captured HTTP packets. Many websites use different versions of their pages for the mobile devices. These pages are suitable to view in smaller screens and serve different advertisements. Also, in mobile devices, third-party applications show ads inside the application window (in-app advertising). However, we only consider malware that are part of a botnet and launch attacks using technologies similar to the desktop malware.

FCFraud detects the fraudulent processes that programmatically click on ads while executing in the background. It is impossible for the fraudulent processes to generate real clicks or touches. These processes either make a new HTTP request to simulate a click on an ad or generate software simulated events which are distinguishable from real events at the operating system level. FCFraud generates alerts and offers users a choice to block internet for the fraudulent processes.

We also design a cloud backend of FCFraud for the smart devices. As some smart devices (e.g., smart phones) have limited power capabilities, FCFraud will provide a choice to send the data to the cloud. After analyzing the data, the cloud will notify FCFraud if any fraudulent process is detected.

In summary, we make the following contributions:

- We take the first step to propose a click-fraud prevention method, FCFraud, on the user-side which can be a valuable addition to the server-side detection techniques.
- We design and implement FCFraud for both desktop and mobile platforms.
- We develop a cloud backend for the FCFraud's mobile version to offload the analysis task to the cloud.

The remainder of the paper is organized as follows. Section 2 provides background knowledge on online advertising, click-fraud and automated clickers. Section 3 describes a number of related work. In Section 4, we present our click-fraud prevention method FCFraud. Implementation details of FCFraud is given in Section 5 and we evaluate it in Section 6. Finally, we summarize the paper in Section 7.

[†]The initial version of this work was published in [13].

DOI: 10.1002/smr

2. BACKGROUND

Nowadays, it is hard not to see an advertising content when we visit any website or use free mobile apps. Publishers not only publish own contents to attract the audience but also sell space on their websites to receive advertising income or for other business promotions. Similarly, mobile developers use advertising to generate revenues. In this section, we first define a number of terminologies related to the online advertising industry. Then, we describe a number of advertising types pertinent to our discussion. Next, to provide a better understanding of the click-fraud attack, we explain how online advertising works and how botnets perform click-fraud.

2.1. Terminology

- *Publisher* is an entity which publishes content or offers a service through a website or a mobile application.
- *Advertiser* is an entity that pays the ad networks for displaying their advertisements.
- *Ad networks* are entities that manage publishers and advertisers. They are able to buy and sell ad traffic (in the form of ad requests) internally as well as through other ad networks. Ad networks that can buy and sell traffic between each other are called linked partners, and each ad network maintains its own list of trusted partner networks.
- *Ad exchanges* facilitate the real time buying and selling of inventory from multiple ad networks. The approach is technology-driven as opposed to the historical approach of negotiating price on media inventory.
- *Impressions* are a metric that counts the number of times an ad has been deployed.
- *Clicks* are another metric that counts the number of times users clicked (using keyboard, mouse, and/or touchscreen) on an ad.
- *Ad request* is the form of HTTP traffic that is generated from impressions or clicks.
- *Ad request response* is the form of HTTP traffic that is generated by the ad networks in response to any ad request.
- *Real-time bidding (RTB)* is a means by which advertising inventory (ad space that a publisher wants to sell to advertisers) is bought and sold on a per-impression basis. It is done via programmatic instantaneous auction similar to stock markets.

2.2. Types of advertising

There are many types of online advertising. For example, display-based advertising, search-based advertising, social media marketing, email advertising, chat advertising, classified advertising, affiliate marketing, and content marketing. In FCFraud, we consider ad traffic generated from display-based, search-based, and mobile advertising only. Below, we describe each of them.

2.2.1. Display-based advertising. Display-based advertising conveys its advertising messages visually using text, logos, animations, videos, photographs, or other graphics. Display advertisers frequently target users with particular traits to increase the ads' effect. Online advertisers (typically through their ad servers) often use cookies, which are unique identifiers of specific computers, to decide which ads to serve to a particular consumer. Cookies can track whether a user left a page without buying anything, so the advertiser can later retarget the user with ads.

2.2.2. Search engine marketing (SEM). Search engine marketing, or SEM, is designed to increase a website's visibility in search engine results. Search engines provide sponsored results and organic (natural or non-sponsored) results based on a web searcher's query. Search engines often employ visual cues to differentiate sponsored results from organic results. Search engine marketing includes all of an advertiser's actions to make a website's listing more prominent for topical keywords.

DOI: 10.1002/smr

2.2.3. Mobile advertising. Advertisements are delivered through mobile devices such as smartphones, tablets or other smart devices (e.g., smart TVs). Mobile advertising may take the form of static or rich media display ads, SMS (Short Message Service) or MMS (Multimedia Messaging Service) ads, mobile search ads, advertising within mobile websites, or ads within mobile applications or games. Mobile advertising is growing rapidly. For example, in January 2016, Facebook reported that their advertising revenue for the September 2015 quarter is $5.841 billion. Of that, mobile advertising revenue accounted for around 80 percent [14].

2.3. How online advertising works

Nowadays, online advertising is a very complicated process which begins when a consumer uses a browser to visit a webpage. The browser opens a connection to the publisher's content server and the server returns the content for the page. Then, the browser requests the ad network to serve an ad. The ad network may connect to an ad exchange with necessary client information for real-time bidding. After an ad is selected, the ad network returns the ad information to the browser. The browser now visits the agency ad server to retrieve the ad (image/rich media) and the ad network records the request as an impression. The browser now renders the actual page with the ad. The ad contains a link to the ad network. When any user clicks on an ad, the ad network first registers the click for billing and then redirects the browser to the advertiser's page. In mobile devices, the same process applies to web browsing. For in-app advertising, mobile apps use SDKs (software development kits, supplied by the ad networks) to connect to the ad network and request for ads. The ad network then returns an ad following the above process.

2.4. Automated clickers or clickbots

Clickbots are special software that can click on online ads automatically. Such clicks are often called "invalid clicks". Invalid clicks are any clicks that an ad network chooses not to charge for. Nowadays, clickbots are very sophisticated and often equipped with the capabilities of a real browser. They crawl to different websites and click on links provided by their botmasters. Some of them can imitate real human browsing behaviors and mouse movements.

A clickbot in a botnet performs some common functions including initiating HTTP requests to a webserver, following redirections, and retrieving contents from a web server under the control of a remote botmaster. A botmaster can leverage 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 uses internet (drive-by download, free mobile apps, etc.) to distribute malware to the victim host. Then, 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 referer to be used in the fabricated HTTP requests, what kinds of ads to click on, and when or how often to click [11]. After receiving instructions, the clickbot begins traversing the designated publisher website and simulates a click on each selected ads. The ad network logs the click traffic and then returns a HTTP 302 redirect response to the advertiser's page. Every time an ad is clicked by a clickbot, the advertiser pays the ad network if it is not detected as "invalid" and the involved publisher receives a portion of the revenue from the ad network.

3. RELATED WORK

Advertising revenue is one of the major sources of income for companies that own ad networks (e.g., Google, Microsoft, and Yahoo). Many independent web publishers also rely on the advertising income to maintain their hardware and software cost. As a result, click-fraud has become a major threat to the ad networks as well as web publishers [15, 16]. For example, a wide range of attacks against the online advertisement industry has been documented in the literature [11, 17, 18, 19, 20]. In many of those attacks, botnets have been used extensively. For example, in 2006, a botnet

DOI: 10.1002/smr

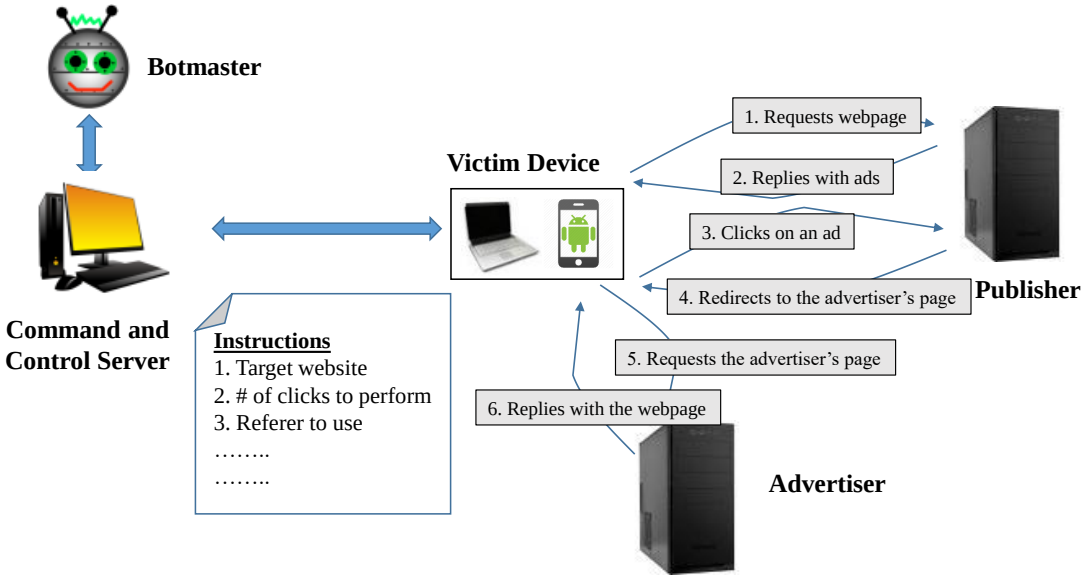

Figure 1. How botnet click-fraud works.

consisting of 115 computers was discovered, which was designed to execute low-frequency click-fraud attacks [20]. Later on, a much larger botnet was discovered by Panda Labs [21] consisting of 100,000 computers. The attacker, in this case, infected the machines using a popular screensaver. Recently, the ZeroAccess botnet was costing US advertisers 2.7 million USD per month when it was active with around 50,000 computers [22].

Due to the increasing nature of the threat, researchers from ad network companies are investigating the problem extensively. Advertisers also have a strong interest to combat all kinds of advertising frauds that are draining their money. Publishers, ad networks or the advertisers use global advertising web traffic and their own proprietary technology to detect fraudulent clicks [23, 24, 25]. For that reason, most research in this field tries to detect click-fraud from the server-side. However, in our understanding, click-fraud is a user-side behavior. Therefore, our approach focuses on the user-side detection of click-fraud. In this section, we systematically discuss the most pertinent literature about click-fraud attacks and detection by cataloging their weakness and strengths. First, we list a number of works that operate from the server-side. Then, we discuss the user-side techniques. A summary table of related work is included at the end of the section.

3.1. Server-side techniques

Metwally et al. [26] presented three forms of click-fraud and the methods to detect them. They found that several websites can cooperate with each other to create fraudulent clicks and thus advance their commercial interests. They developed an algorithm, called *Streaming-Rules*, to detect fraud in advertising networks. The main idea of the algorithm is to find associations between elements in a stream of HTTP requests from customers and detect fraudulent patterns. The authors claimed that *Streaming-Rules* reports association rules with tight guarantees on errors and uses minimal space. Later, in [27], they generalized the solution to detect a variety of complex coalition attacks using their similarity-seeker algorithm. These techniques offer more scalability than traditional techniques by including statistical data analysis on web traffic. However, these approaches apply only to botnets and malware-driven click-fraud. They cannot detect click-fraud performed by real humans.

Immorlica et al. [28] studied fraudulent clicks and presented a click-fraud resistant method for learning the click through rate (CTR) of advertisements. Their method works in a system where each advertiser submits a bid which is the maximum amount they are willing to pay per click. The advertisers are then ranked based on the product of their bids and respective estimated CTRs of

DOI: 10.1002/smr

their ads. The CTR is defined as the likelihood, or probability, that an impression of an ad generates a click. A fraudulent click in such a system causes short-term loss for the advertisers. Moreover, a fraudulent click will be interpreted as an increased likelihood of a future click resulting in an increase in the estimate of CTR. The authors proved that their CTR learning algorithm can cancel short-term loss and long term benefit of a fraudulent click. Similarly, Dave et al. [8] presented an approach which is designed based on the invariant that click-spam is a business (for click-spammers) that needs to deliver high return on investment (ROI) to offset the risk of getting caught.

Haddadi [9] suggested that advertisers can use bluff ads to detect fraudulent clicks on their ads. Bluff ads are either targeted ads, with irrelevant display text, or highly relevant display text, with irrelevant targeting information. They are designed as a test for the legitimacy of the individual user clicking on the ads. While bluff ads may be effective in detecting click-fraud to some extent, advertisers have to spend extra money on those bluff ads. Also, for the publishers, it is not practical to show meaningless ads to the real users.

Xu et al. [29] proposed a new approach for advertisers to independently detect click-fraud activities issued by clickbots and human clickers. Their 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, the system observes user behavior on the advertised site. However, both of these approaches are easily bypassed by today's sophisticated malware that imitate human behavior and use full-fledged browser engines.

3.2. User-side techniques and comparison with FCFraud

Researchers [30, 31] begin to investigate user-side ad behavior in the advent of mobile advertising. Crussell et al. [32] analyzed Android malware apps to investigate ad-related frauds. They used a total of 165,426 Android apps and executed them in an emulator without any user inputs. They collected the network packets generated by each app to extract ad requests and ad clicks from the traffic. Their analysis showed that click-fraud is also a legitimate threat in the mobile advertising industry. However, they did not provide any measure to prevent click-fraud. In our work, we designed FCFraud to prevent click-fraud by making it a part of the operating system in the user machines.

Recently, Cho et al. [33] proposed a method to distinguish between real and software generated touch events to combat click-fraud in mobile devices. They modified Android to block all simulated events in the operating system. Although, the method can successfully block simulated clicks, it will fail to thwart clicks that make use of the HTTP techniques. Also, in our opinion, simulated events are sometimes necessary and blocking all simulated events may hamper functionalities of certain types of applications. The major difference between their work and FCFraud is that we detect fraudulent processes by monitoring their internet activity. Then, users can block the processes from accessing the network. FCFraud does not interfere with the operating system events.

3.3. Summary of the related work

Click-fraud is regarded as the number one problem in the online advertising industry. Many research papers documented the problem. In addition, research has been done to detect and/or prevent click-fraud from the server-side. A few papers also proposed theoretical model to measure the click traffic quality. In Table I, we summarize the research works described in this section. We also include the type of data and methods used in each technique. In comparison with other user-side techniques, FCFraud is the only technique that works in both desktop and mobile platforms.

DOI: 10.1002/smr

Table I. Summary of related work.

Research	Server/ User-side	Desktop/ Mobile	Data	Method
Metwally et al. [2005,2007] [26, 27]	Server	N/A	Web traffic	Statistical data analysis
Immorlica et al. [2005], [28]	Server	N/A	Click traffic	Automatic learning
Dave et al. [2013] [8]	Server	N/A	Click traffic	Bayesian framework
Haddadi [2010] [9]	Server	N/A	Web traffic	Displaying bluff ads
Xu et al. [2014] [29]	Server	N/A	Browser behavior, click traffic	Javascript capability testing and machine learning
Crussell et al. [2014] [32]	User	Mobile	Web traffic	Machine learning
Cho et al. [2016] [33]	User	Mobile	Operating system events	Operating system hooks
FCFraud	User	Both	Web traffic and Operating system events	Operating system hooks and machine learning

4. APPROACH OF FCFRAUD

We design FCFraud as a part of the user operating system to prevent click-fraud[‡]. FCFraud tries to detect programs that execute in the background, implement browser functionalities and perform click-fraud stealthily. We present the workflow of FCFraud in Figure 2. FCFraud has two modules, namely, the inspection module and the analysis module. The inspection module continuously captures HTTP packets and input events and stores them in a database. Periodically, the analysis module extracts HTTP packet and event information per process from the database and tries to find whether any process is fraudulent. First, it creates HTTP request trees for each process. It then generates features from the request trees and classifies the HTTP packets to identify ad requests. Next, it identifies ad clicks in the request trees. The analysis module marks a process fraudulent if the process does not generate real input events, however, its tree contains ad requests and clicks. By real events, we mean the events that are generated by the hardware input devices (e.g., mouse, keyboard, and touch screen), not software generated simulated events. Processes that interact with a real user are marked non-fraudulent. Nonetheless, FCFraud includes all processes in the report that perform ad-related activity. We implement FCFraud for a debian-based Linux distribution (the

[‡]The description of different modules of FCFraud is applicable to both desktop and mobile versions unless stated explicitly.

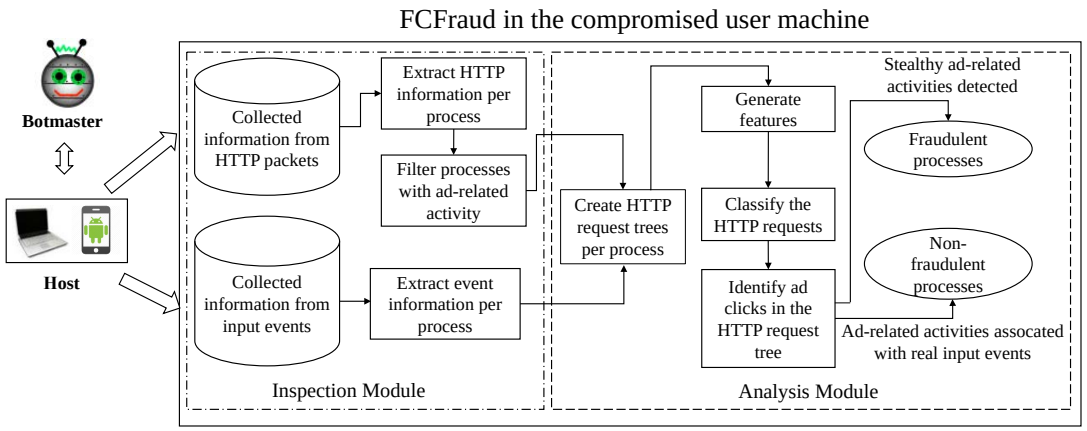

Figure 2. Workflow of FCFraud.

DOI: 10.1002/smr

```
while Machine is ON do
      -Capture HTTP Packets
      -Capture input events
      -Apply ad blocking filter to the HTTP requests
      -Identify processes associated with ad-related traffic
   for each Processes with ad-related traffic do
      -Create HTTP request trees
      -Generate features and classify the HTTP requests
   end for
   -Detect and block the fraudulent processes
end while
```

Figure 3. Click-fraud prevention algorithm.

desktop version) and the Linux-based Android operating system (the mobile version). We also implement a cloud backend for the mobile version as some mobile devices have resource limitations. In a nutshell, the steps of FCFraud are shown in Figure 3 and we describe each step in detail in the following subsections.

4.1. Capture and collect information from HTTP requests and input events

FCFraud captures packets from all network interfaces and analyzes headers of the packets. If the captured packets contain HTTP requests or responses, FCFraud reads the headers and collects information. FCFraud uses the libpcap [34] packet capture library to capture network packets. In Figure 4, we demonstrate how FCFraud collects information from each running process.

In Linux, captured network packets do not have any information on the originating process. As a result, FCFraud looks into the "/proc" file system of the Linux kernel to find out the process which is currently using the source port of the captured packet. Once a process is found, FCFraud records different fields from the HTTP request and response along with the process name. To find the response of a request, it matches the source port of the request with the destination port of the response and vice versa. From the request, it extracts header fields like timestamp, source ip, source port, destination ip, destination port, referer as well as the host and the request URI (uniform resource identifier). From the response, it records location, content type, content length, content

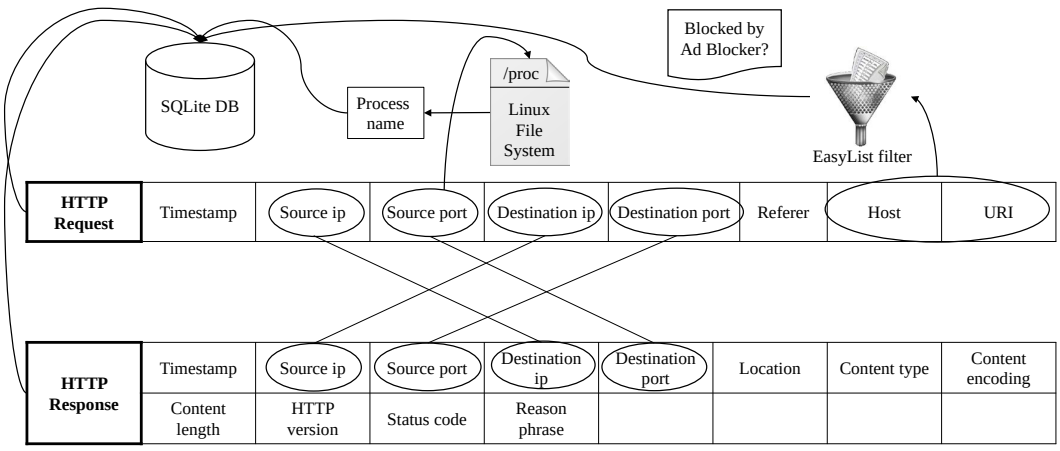

Figure 4. Capturing HTTP packets and extracting information per process.

DOI: 10.1002/smr

encoding, HTTP version, status code, and reason phrase. FCFraud stores all these information in a database for later steps. At this time, FCFraud does not inspect the body of the requests. Also, it does not save the packets for later analysis as it may require gigabytes of storage space.

In desktop, while capturing network packets, FCFraud also records the timestamp of each mouse/keyboard event received by a process. It records events from hardware devices (e.g., mouse and touchpad) only. In Linux, input events are gathered at the `/dev/input/event#` file. FCFraud executes separate threads for each input device present in the system. At the time of recording, it determines the top window id of the x-window system and finds all the process ids (pid) associated with the window. It saves this information as "event|pid1, pid2, pid3, ...|time" in the database. This information is required to identify processes that execute in the background (no visible windows) and do not generate any real input events (e.g., mouse clicks).

In the mobile version, the event capturing process is similar to the desktop. However, in Android, only one application can be active at any given time. As a result, we store an event, its occurrence time and the process id and user id (uid) of the active app at that time. This information is stored as "event|pid, uid|time" in the database. From this information, FCFraud knows whether a process is the active one when the process is making an ad-related HTTP request.

4.1.1. Apply ad blocking filter and filter processes. FCFraud uses an ad blocking filter to decide whether an application is doing ad-related activity. Ad blocking filters are rules used by popular ad blockers (web browser extensions/apps that block ads). They can filter ad-related contents via URL filters, DOM element filters, and third-party advertisement domain filters. FCFraud uses EasyList [35] (46,423 rules) for both the versions. Easylist is free, open-source, and maintained by the online community. After saving HTTP packet information, FCFraud checks each of the URL against all these rules and saves the decision (ad-related and not ad-related) in the database. This decision is also used as a feature in the ad request classifier later on. Based on this decision, FCFraud filters processes to be analyzed further for fraudulent ad-related activities. This filtering process saves unnecessary computational power for the later steps of FCFraud.

4.2. Create the HTTP request trees

After the filtering of processes with ad-related traffic, FCFraud generates HTTP request trees for each of such process. The idea of the HTTP request tree comes from the fact that HTTP requests of the processes can be grouped together logically. The browser may send a thousand requests just to load one web page. For example, a browser loading a web page may fetch many other static resources, such as CSS, JavaScript or images, to embed in the HTML. In this case, we can group the HTTP requests using the HTTP referer header to form a tree of requests where the request to the HTML page is the root and the requests to the static resources are the children. In this way we can separate the request to a new web page from requests that are used to load contents of an already requested web page. If we can construct a tree that represents the exact scenario of the browsing (showing all the visited pages as nodes), then it will be easier to analyze and automatically detect ad clicks. We use the FCFraud database to create the HTTP trees. In Table II, we present a partial view of the data stored in the FCFraud database when loading `www.foxnews.com`. The table shows the values of source port, destination port, direction, host (domain name in the HTTP packet) and

Table II. A partial view of the HTTP capture database.

#	Source process	S-port	Dest-port	Direction	Host	Referer
1	Firefox	43500	80	>	foxnews.com	-
2	Firefox	43643	80	>	ads.foxnews.com	foxnews.com
3	Firefox	43648	80	>	global.fncstatic.com	foxnews.com
4	Firefox	43649	80	>	global.fncstatic.com	foxnews.com
5	Firefox	80	43500	<	-	-

DOI: 10.1002/smr

referer. The direction field indicates whether the packet is a part of a request or a response (e.g., Row 5 is the response of Row 1).

We represent each HTTP request and its corresponding response as a single node in the request tree (Row 1 and 5 in Table II form a single node) and connect two nodes if:

1. The latter node (Row 2,3, and 4 in Table II) contains the request referer field set to the host of the former node (Row 1 and 5 in Table II). We consider the former node as the parent of the latter and mark the edge as "REFERER".

2. The former node contains the location header along with a redirection status code to redirect the client to the latter node. We consider the former node as the parent of the redirected node and mark the edge as "LOCATION".

3. The latter node contains the client id of the former node in its URL. We consider the former node as the parent node of the latter and mark the edge as "CLIENTID". In this case, we use the unique client id of a publisher which is assigned by the ad network at the time of registration.

Each process may have multiple trees associated with them. We show an example of HTTP request trees of a process connecting two hosts in Figure 5. The edges are marked as "REFERER" and "LOCATION" based on the above-described rules. We describe the figure in details in subsection 4.4.

4.3. Generate features and classify the HTTP requests

To detect ad-related fraud, we must first identify ad requests in the HTTP request trees. There are many kinds of ad providers in the web space and it is very difficult to identify all kinds of ad requests by some hard-coded rules. FCFraud uses machine learning classification to automatically identify ad requests. The results of machine learning algorithms are often much more accurate than human-crafted rules. Also, in this context, the formats of the web advertising links are not standardized and they may change over time. Therefore, machine learning enables us to retrain the classifiers automatically.

Web ad requests have some common characteristics. For example, they normally have a large number of query parameters and their responses are normally images or JavaScript contents. FCFraud extracts features from the query parameters, the HTTP headers (both the request and the response), and the constructed HTTP request trees. To make the classification more accurate, it also uses the decision of an ad blocking filter as described in Section 4.1.

We classify the URLs into two classes, `ad-related` and `not ad-related`. Ad-related URLs are requests to an ad network and not ad-related URLs are all other requests.

4.3.1. Features. To make the classifier effective, we try to find unique features that may help greatly to differentiate an ad-related URL from a non ad-related URL. These include characteristics of the structure of the URL, other URLs present in the URL, and different page properties. All features are either binary or integer and given equal weight at the training time. Below, we describe a summary of the key feature categories used to train the classifiers.

Features from the HTTP packet headers. The features in this set are from the inspected HTTP packet headers. This set includes destination ip, content type, content length, status code, location URL, etc. From the location URL, we calculate the number of subdomains in the link and its length. We also create features from the host name and the request URI. Some examples are the length, the number of subdomains, whether a referer is present or not, and if present, the length and the number of subdomains in the referer.

Features from the query parameters. Query parameters are very important in our case. A typical ad-related HTTP request contains a large number of query parameters. These parameters send many

DOI: 10.1002/smr

information about the particular client machine to find a matching ad for the request. The number of query parameters, the average length of the parameters, etc. are examples of features from this set.

Features from the HTTP request trees. FCFraud constructs the request trees to recreate the browsing scenario. For this set, we select the height of the subtree rooted at each node, the number of blocked URLs (by the ad-blocking filters) in the subtree, the number of images and JavaScript requests in the subtree, the number of different domains in the subtree, etc.

4.4. Detect and block the fraudulent processes

To detect ad fraud, FCFraud analyzes the HTTP request trees and collected input events. Then, it tries to detect ad clicks in the HTTP request trees. Based on the detected ad requests and ad clicks along with the input events database, FCFraud marks a process fraudulent.

Ad click detection is one of the trickiest task of FCFraud. When a user clicks on an ad, the browser generates a HTTP request to the ad network, the ad network records the event and then redirects the browser to the advertiser's page. The address of the advertiser's page is typically provided in the location header of the response. Therefore, in our request trees, we mark a node as ad click if it is a child of an ad request, the edge is marked as "LOCATION" and it is the root of a subtree whose nodes represent a separate website visit. When there is no location header, we consider nodes that are roots of subtrees representing separate website visits. If the root of such a node contains a pre-specified number of ad request nodes in its tree, then it is an ad click. In this case, we assume that the root contains a number of ads and the subtree is formed because of the user clicked on one of the ads. In our experiment, we find that these nodes normally use the former pages as referer.

We present both the scenarios in Figure 5. The browser process "Process 1" goes to `www.hosta.com` which contains an ad (the ad request node). The user clicks on the ad and visits `advertiser.com/index.html`. Here, these two nodes are connected by a "LOCATION" edge and the advertiser's page is a different website (not in the `www.hosta.com` domain). As a result, we mark the advertiser node as "ad click". In the second scenario, `www.hostb.com` contains a number of advertises in the page and the node `advertiser.com/index.html` represents a different website visit. It is also marked as "ad click".

Finally, we execute a filter to remove false-positives from the result. Websites load contents from different domains and that node in the HTTP request tree can be falsely detected as an ad click. We also observe that many requests to the ad networks, web analytics, and web tracking sites generate further communications with them and those later requests use the first one as the referer, which are also falsely detected as ad clicks. Additionally, we notice that some of the requests are made

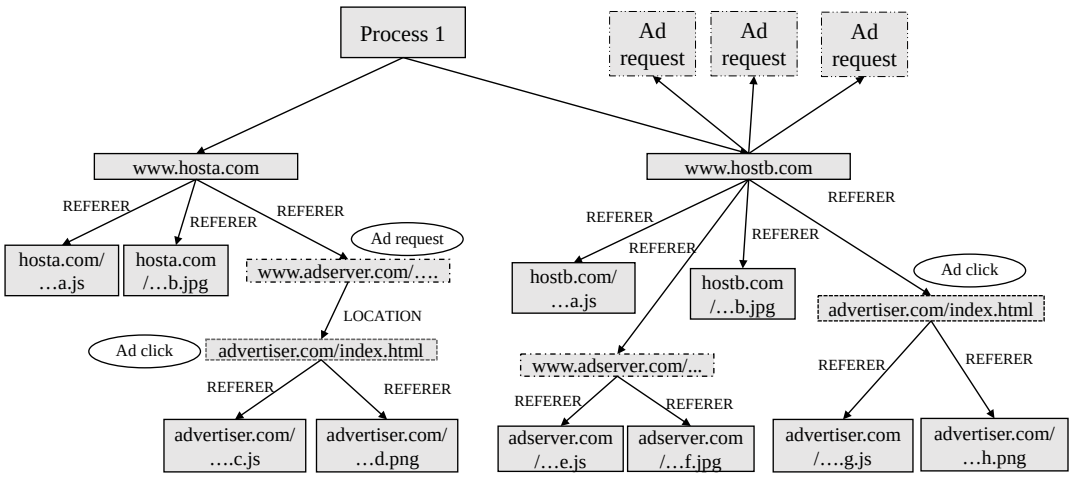

Figure 5. An example of HTTP request trees of a process. We encircle the ad click and ad request nodes.

DOI: 10.1002/smr

Classifier: notad **Suspicious**: yes **isAdClick**: yes **NumberOfAdinChild**: -1 **SrcP**: 43642
GET: `global.fncstatic.com/static/v/fn-hp/css/ag.home.css?20141021T1229`
Ref: `www.foxnews.com/`
ResponseLocationHeader: `http://global.fncstatic.com/static/v/fn-hp/css/ag.home.css?20141021T1229`
EdgeType: LOCATION

Classifier: notad **Suspicious**: yes **isAdClick**: yes **NumberOfAdinChild**: -1 **SrcP**: 48773
GET: `www.viarail.ca/sites/all/files/css/css_fd24a8a78a15b3f0afb2d5b475550379.css`
Ref: `www.viarail.ca/en/train-advantages/?utm_campaign=_14q1nonb&utm_medium=_ban&utm_source=_accutab&utm_term=_ros_300 x250_en_150126&utm_content=_kin-tor$41`
ResponseLocationHeader: -
EdgeType: REFERER

Classifier: notad **Suspicious**: yes **isAdClick**: yes **NumberOfAdinChild**: -1 **SrcP**: 53841
GET: `player.cnevids.com/embed/5480d98f61646d5d5d040000/51799c3f68f9da5d48000002`
Ref: `www.wired.com/`
ResponseLocationHeader: -
EdgeType: REFERER

Figure 6. Examples of false-positives in ad click detection.

to static resources, such as CSS files which can be removed from the list without hampering the performance of the ad click detection. Figure 6 demonstrates a number of such false-positives. In each of the three cases, the request URI (obtained from the value of **GET** in the figure) corresponds to a static resource file.

After the ad click detection, FCFraud marks a process as fraudulent if the process clicks on an ad using a software simulated click or an independent HTTP request (without a click) while executing in the background. It uses the input event database to determine whether this particular process generated any real events (i.e., a real user interacts with it). If FCFraud identifies any fraudulent process, it notifies the user and offers the user a choice to block the process from accessing the network. To block network access, FCFraud uses the time module of the iptables [36] tool.

5. IMPLEMENTATION

To evaluate the effectiveness of FCFraud, we need to execute clickbots and FCFraud simultaneously. For that reason, we implement FCFraud and a number of clickbots for both desktop (Ubuntu Linux) and mobile (Android) systems. In both versions, FCFraud has two modules, namely, the inspection module and the analysis module. The inspection module executes as a daemon service with root privileges. The service is written in C and communicates with the analysis module that executes in the user space. The service inspects HTTP requests and saves necessary information along with the input events in an SQLite database. We implement the mobile version of the inspection module by modifying the Android Open Source Project (version 4.4). In Android, the inspection module is a system service which starts with the device `init`. We also build an app to control the inspection module in mobile devices.

The analysis module analyzes the database of HTTP requests, detects fraudulent activities and alarms the user periodically. In the desktop version, the analysis module is programmed in Java. We use WEKA machine learning library [37] to classify HTTP requests. In the mobile version, this module is implemented as a part of the FCFraud app that communicates with the system service

DOI: 10.1002/smr

component. They communicate using the binder [38] IPC mechanism. The FCFraud mobile app is a standard Android app written in Java. It gives the user control over the inspection module and the cloud analysis feature.

The details of the classifier, cloud back end, and click bots are given below.

5.1. Select the right classifier.

To select the best classifier for FCFraud, we test five classification algorithms, namely, Naive Bayes, Support Vector Machines (SVM), K-Nearest Neighbors, C4.5 [39], and Random Forest [40]. To evaluate the average classification accuracy of each classifier, we use the k-fold cross validation (CV) with k=3. k-fold CV ensures that the classifier is trained on every example. In comparing the effectiveness of the classifiers, we look at the average accuracy of the testing data as well as the precision and the false positive rate. In our case, we like to minimize the false positive rate and maximize the recall of the positive ("ad") class. The formulas for calculating the accuracy, precision, false positive rate, and recall are given below:

$$accuracy = \frac{true\ positives + true\ negatives}{true\ positives + false\ positives + false\ negatives + true\ negatives}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$false\ positive\ rate = \frac{false\ positives}{false\ positives + true\ negatives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

5.2. Click-fraud detection at cloud for mobile devices.

Although, mobile devices are becoming powerful day by day, they are still not ideal to perform cpu-intensive analysis because of the power consumption overhead. We design and implement a feature for FCFraud where a user can choose to perform the click-fraud detection in the cloud. Using the FCFraud app, users can enable the cloud detection feature. The app uses web service technology to send the data to the cloud in an encrypted manner. After obtaining the data, the cloud server executes the Java program that we use in the desktop version to analyze the data. FCFraud's database contains only textual information and can be easily sent to the cloud.

The cloud backend has three modules, namely, **authentication**, **click-fraud detection**, and **logging and reporting**. First, we have the authentication module. It authenticates the FCFraud app so that the app can continue communicating with the cloud. The authentication module uses the device credentials and a password to authenticate a device. Second, there is a click-fraud detection module that analyzes the information in the database sent from the mobile devices. Using the FCFraud app, a user will be able to send the database and receive the analysis report. Users can also block/unblock internet for the suspicious applications using the app. Third, there is a logging and reporting module that logs all activities of the app. FCFraud's service components are also programmed to send periodic status messages to the logging system.

5.3. Click bots.

To examine the effectiveness of FCFraud, we implement a number of clickbots for both the desktop and mobile versions. In the desktop version, we use three kinds of bots to generate the fraudulent traffic. Two of them use browser drivers to control Google chrome and PhantomJS [41]. Browser drivers or web drivers are application programming interfaces (API) that can communicate with a browser and perform all the user interactions programmatically. In our experiment, we use the Selenium WebDriver [42]. Google chrome creates a visible window and thus noticeable by the user. As a result, in case of Google chrome, we make the window hidden by using the Xdotool [43]. PhantomJS is a headless browser and does not create any human viewable window. The third bot

DOI: 10.1002/smr

is an independent program, which generates HTTP traffic and parses the response. We use a web browser control to implement the feature.

In the mobile version, we create two Android applications. One application is a simple number game and another application is a movie information finder. We include click-fraud capability in both the applications. Both the apps require internet permission. The apps visit webpages specified in the instruction and click on ads programmatically.

We develop another program that acts as a botmaster. It communicates with the bots using the HTTP protocol. The botmaster supplies website addresses and XPaths of the web elements containing ads to the bots. XPath is a query language for selecting nodes in an XML document. The bots first visit the websites given to them. Then, they generate clicks on the web elements containing the ads. Occasionally, the bots generate multiple clicks on the same page if multiple XPaths are supplied to them.

6. EXPERIMENTAL EVALUATION

In this section, first, we describe the devices we used in our experiments with configurations. Then, we describe how we create our ground truth dataset to evaluate different classifiers. We also present the result of the experiment for selecting the appropriate classifier for FCFraud using real ad traffic (the ground truth dataset). After that, we observe the effectiveness of FCFraud based on how many clickbots it detects. We implement three clickbots for the desktop system and two clickbots for the mobile system to generate fraudulent traffic. We conclude the section with an illustration on the operational overhead of FCFraud in mobile devices.

6.1. Experimental environment

In our experiment, for the desktop version, we use a Pentium Core i7-4770 3.4GHz machine with 16GB of RAM and Ubuntu 14.04 64-bit operating system. For the mobile version, we use ODROID XU3 [44] as our experiment device. ODROID XU3 is an ARM device from the company Hardkernel[45] and it has Samsung Exynos 5422 Cortex-A15 2.0 GHz quad-core CPU, Mali-T628 MP6 GPU, and 2 GB DDR3 SDRAM. To implement the cloud detection feature, we select Microsoft Azure as our cloud provider. We use a PAAS Instance (Model D2: 2 Core, 8GB RAM, 100GB SSD, Windows Server 2012) and an Azure SQL DB Instance (2 TB).

6.2. Ground truth for the ad request classifiers

To experiment with different classifiers and choose the best one for our needs, we create a ground truth dataset. We select 44 websites (listed in Table III) containing advertisements for the desktop and mobile version. We try to pick the most popular categories (from `www.alexa.com`) such as arts, business, health, science, shopping, and sports. As mentioned before, we only consider clickbots in mobile that use the same desktop techniques to launch attacks. However, to show the effectiveness of our classifier on the ad library generated ad traffic, we also build a ground truth dataset by executing 25 mobile apps. The apps chosen are mostly the mobile counterparts of the websites selected. Table IV lists all the apps that are used in this step.

To build the dataset, we visit all the websites using the Firefox web browser and inspect the HTTP requests using tcpdump [46]. We manually classify 1,629 ad requests from the 13,358 HTTP requests generated by Firefox. In mobile, we run all the apps and click on ads where possible. We compiled a version of the tcpdump for Android using the arm compiler to inspect the HTTP packets in the mobile. We classify 87 ad requests from the 1,754 requests generated by the apps.

6.3. Experiments and results

To detect click-fraud, we need an appropriate ad request classifier for FCFraud. The classifier should identify most of the ad requests (maximize the recall) for our ad click detection. At the same time, it needs to be accurate enough (low false positives). To select the best classifier, we experiment and

DOI: 10.1002/smr

Table III. List of websites selected for the creation of the ground truth data.

#	Category	Website	#	Category	Website
1	Arts	popsugar.com	8	News	foxnews.com
		rottentomatoes.com			reddit.com
		azlyrics.com			huffingtonpost.ca
		wired.com			theweathernetwork.com
2	Business	ca.eonline.com			accuweather.com
		wsj.com	9	Recreation	homeaway.com
		ibtimes.com			autoblog.com
3	Computers	about.com			cracked.com
		ask.com			lonelyplanet.com
		mashable.com			skyscanner.ca
4	Games	cheatcodes.com	10	Reference	urbandictionary.com
		ca.ign.com			stackoverflow.com
		gamefaqs.com			wordreference.com
5	Health	earthclinic.com	11	Science	sciencedirect.com
		myfitnesspal.com			howstuffworks.com
		mensfitness.com			pcworld.com
6	Home	gizmodo.com	12	Shopping	newegg.com
		gsmarena.com			kijiji.ca
		allrecipes.com			cars.com
7	Kids and Teens	sciencedaily.com	13	Society	salon.com
		mathsisfun.com			slate.com
		rhymezone.com			snopes.com

calculate the effectiveness of the classifiers on the ground truth dataset (both browser and ad library generated). Next, we show that FCFraud is indeed very effective as it detects all the clickbots.

6.3.1. Select the appropriate classifier for FCFraud's analysis module. We use the selected classifiers (described in Section 5.1) to classify the HTTP requests in our ground truth dataset. The classifiers classify a request as either "ad" or "notad". We consider the instances from the class "ad" as positive. Table V shows the comparison of the average accuracy, the precision and the false positive rate of the various classifiers. In Figure 7, we report the recall of different classifiers.

In the desktop ground truth data, RandomForest has the highest average accuracy and precision of 99.61% and 98.05%. It also has the lowest false positive rate of 0.27%. Though the average accuracy, precision and false positive rate of the SVM algorithm is promising, it has a poor recall value of only 62.98% as shown in Figure 7. RandomForest produces the highest recall value of 98.77%. However, in our view, all the algorithms show satisfactory results. We believe that the acceptable performance of the classification algorithms comes from the choice of the features and the use of the decision of an ad blocking filter. In conclusion, we decide to use RandomForest as the classification algorithm in FCFraud.

In the mobile version, 5kNN produces the highest average accuracy of 99.28%. RandromForest has the second best value of 98.23% in terms of accuracy. Table VI presents the confusion matrix

Table IV. List of mobile apps selected for the creation of the ground truth data.

#	Mobile Apps	#	Mobile Apps
1	Fox news	14	Cracked.com
2	Urban Dictionary	15	Lonely planet
3	The weather network	16	Trip advisor
4	Accuweather	17	Skyscanner
5	Salon.com	18	Home away
6	Stack exchange	19	CNN
7	Science daily	20	BBC
8	Wikihow	21	Zomato
9	Askfm	22	Restaurant Story: Hot Rod Cafe
10	Answers.com	23	Bubble Shooter Games
11	Huffingtonpost	24	Rapala Fishing - Daily Catch
12	Mashable	25	Heart Live Wallpaper lite
13	Engadget		

DOI: 10.1002/smr

of the RandomForest classifier for both versions of the ground truth data. Overall, we observe that all the classifiers are less effective on the mobile dataset. This is due to the fact that ad libraries used in in-app advertising often use their own format of communication which do not conform with the web ad format. However, FCFraud is developed mainly to thwart stealthy malware that perform click-fraud in the background. These kinds of malware mostly use the web technology to perform click-fraud.

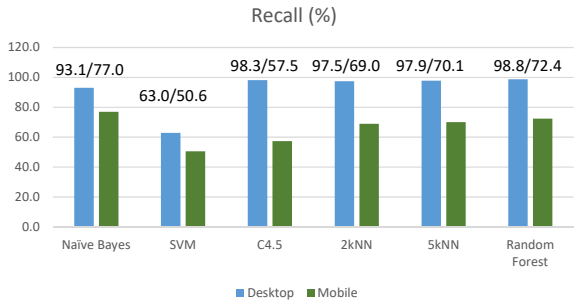

Figure 7. Accuracy of the positive class (recall) of different classifiers (desktop/mobile).

6.3.2. Effectiveness of FCFraud against clickbots. In this experiment, we enable FCFraud and execute our clickbots in the desktop and mobile versions. We execute the bots both sequentially and concurrently to examine the effectiveness of FCFraud in different scenarios. The clickbots communicate with the implemented botmaster program (executing in a machine in the local network) and visit the websites provided by the botmaster. Then, they silently click on ads. In this case, the mobile clickbots visit the same webpages that the desktop clickbots visit. It is worth mentioning that clickbots normally do not use ad libraries as they require verification from the corresponding ad networks. Nonetheless, we already showed in the previous subsection that our classifier performs acceptably for both browser and ad library generated ad traffic.

FCFraud detects 43 ad clicks among a total of 54 (in both desktop and mobile). It could not detect 11 ad clicks because those requests do not contain a referer or location header in the corresponding HTTP packets. Those clicks are likely generated by a dynamic JavaScript code with the referer or the location header hidden in different URL parameters. There are a total of 35 false positives. However, in this step, the most important thing is to detect real ad clicks. False positives do not impact the detection of background fraudulent processes. The recall and precision of the ad click detection are 79.62% and 55.12%.

After the ad click detection, FCFraud successfully identifies all the five clickbots as fraudulent processes in both desktop and mobile. Once FCFraud identifies any fraudulent process, it notifies the user about the incident. The user can block the clickbots from accessing the network using FCFraud if they want. Without network access, the processes become ineffective for the attack as they cannot communicate with the botmaster.

Table V. Performance of different machine learning classifiers to classify ad requests over 3-fold cross-validation. For each classification algorithm, we report the average accuracy and precision and false positive rate of the "ad" (positive) class (desktop / mobile).

Classification Algorithm	Avg. Accuracy (%)	Precision (%)	FP Rate (%)
NaiveBayes	89.76 / 92.99	54.71 / 39.41	10.71 / 6.18
SVM	95.49 / 97.55	100.00 / 100	0.00 / 0.00
C4.5	99.32 / 97.21	96.21 / 80.65	0.54 / 0.72
2kNN	99.27 / 98.18	96.54 / 92.31	0.49 / 0.30
5kNN	99.33 / 99.29	96.72 / 87.14	0.46 / 0.54
RandomForest	99.61 / 98.23	98.05 / 90.00	0.27 / 0.42

DOI: 10.1002/smr

Table VI. Confusion matrix of our RandomForest classifier (desktop / mobile), computed using 3-fold cross validation.

	notad	ad	Recall (%)
notad	11,697 / 1,660	32 / 7	99.83 / 99.6
ad	20 / 24	1,609 / 63	98.05 / 72.4
Precision (%)	99.72 / 98.6	98.77 / 90	

Here, we find that all the ad clicks generated by the clickbots are detected as fraudulent as our bots perform the attacks in the background and do not generate any real events. Also, it does not make any difference whether we execute the bots concurrently or sequentially. In both the cases, FCFraud is equally effective. Another issue is the timing of the click-fraud attack. Some malware tries to generate fraudulent clicks when the computer is being actively used by a user to make the detection of bot generated traffic harder from the server-side. However, FCFruad detects such malware whether they execute the attack in a busy time period or not. As long as the malware executes in the background without generating real events, FCFraud will detect it and alert the user. Some malware create multiple processes and frequently change their executables. In both the cases, FCFraud can successfully detect and block them as it uses iptables rules to block internet access. Overall, our experiments show that FCFraud is effective and implementing FCFraud as a part of the operating system can benefit the advertisers. Also, FCFraud and the server-side detection can complement each other to fight click-fraud.

6.4. Operational overheads of FCFraud in mobile

FCFraud inspects the HTTP packets continuously and stores the information in the database. Then, the analysis module analyzes the information periodically. In our experiment, the size of the FCFruad database was 4.3 megabytes. In the desktop version, FCFraud performs the analysis in the same machine. It only took 15.8 minutes to analyze the ad traffics in the desktop machine. However, in the mobile version, the time is much longer (73.8 minutes). If a cpu-intensive process executes in the mobile device for one hour, it can adversely affect the battery life. That is why, in the mobile platform, users can choose to perform the analysis in the cloud. Alternatively, FCFraud can perform the analysis in the mobile while it is being charged to save power. The analysis time is reasonable in cloud (23.5 minutes). This justifies the development of the cloud backend for the mobile systems.

6.4.1. Overhead of the inspection module in the mobile version. The inspection module is a system service and executes all the time in the mobile version. As mobile devices are limited in resources, we evaluate the overhead of the inspection module in terms of performance, power consumption and disk usage. In each case, we show that there is very little to negligible overhead of using the module.

Performance. We quantify performance using the AnTuTu benchmarking app available from the Android stores. The app tests CPU and memory performance, 2D/3D graphics, Disk I/O, Multitasking, etc and provides a score for each of them. The overall score is the addition of each individual score. The scores do not mean much on their own. However, they can be useful for comparing different devices. For example, if a device's score is 20,000, another device with a score of 40,000 is about twice as fast.

In our experiemnt, the benchmarking app runs concurrently with the standard set of Android apps that launched at boot. Based on the official Android source code, these apps are launcher, contacts (and its provider process), photo gallery, dialer, MMS, and settings. Vendors customize this list and add more apps. On our ODROID XU3, there were a total of 23 apps (including standard apps) in the device when we execute the benchmarking app. We do not kill any pre-loaded apps.

First, we disable FCFraud and get the scores from the app. Then, we compare them with the scores from the app when FCFraud is running. We find no significant difference in the test scores

DOI: 10.1002/smr

(49,683 vs 49,842). Table VII shows the comparison of scores resulted from the benchmarking app. It is clear from these scores that the overall performance is not hampered by activating FCFraud's inspection module.

Table VII. Individual test scores from the AnTuTu benchmarking app.

Test Group	Score	
	FCFraud Disabled	FCFraud Enabled
CPU	21,932	22,017
RAM	4,992	4,943
UX	12,005	11,865
3D	10,754	11,017
Total	49,683	49,842

Power consumption. As smartphones are limited in battery capacity, we measure the overhead of FCFraud in terms of power consumption using the ODROID smart power [47], which is a dc power source. However, it has a data output port that provides the consumed power. We measure the power consumption by running the device with and without the inspection module for one hour. During this time, we continuously browse different webpages and click on ads. The total power consumed when FCFraud in enabled is 1,302 compared to 1,085 (when disabled) in terms of milliampere-hour (mAh). The difference in the measured power is within an acceptable limit in our understanding. Nowadays, smartphones are often equipped with a 3,000 mAh battery and FCFraud will not hamper the battery life of an average user.

Disk usage. FCFraud extracts information from the captured HTTP packets and store them in the database in a textual format. After the analysis task, FCFraud refreshes the database. In any case, the database size is always in the megabytes range in our experiments. As a result, we can say that there is no disk usage overhead of using FCFraud.

6.5. Discussion and limitations

We observe very limited research effort that include click-fraud prevention in user machines. End users do not have any incentives to prevent click-fraud. They do not care if their machines are being used as attackers as long as the malware do not harm their devices. As a result, it is very difficult to motivate them to install a particular software for the benefit of the online advertising industry. Another option is to include the technique as a part of the operating system as proposed in this work. We chose this option as operating system vendors (e.g., Google, Microsoft) often own or operate ad networks and they have strong incentives to include FCFraud. The problem with this technique is the privacy of the user. To effectively detect fraudulent activities, the operating system monitors the web traffic which may not be acceptable by the end users. To our defense, we argue that the battle between security and privacy is not new. Newer operating systems like Windows 10 already monitors a lot of activities and send the data to the cloud. If we can effectively anonymize the data, many user-side techniques can be embedded in the user operating systems crippling millions of devices as bots and greatly reducing the risk of attacks like click-fraud, spam and DDoS. Another advantage of being part of an operating system is that the malware cannot disable FCFraud. Trying to disable it will crash the entire operating system making the machine unusable for the malware.

Due to the lack of prior work that includes click-fraud protection in user side, we could not compare our solution to other techniques. However, we have a strong belief that a solution like FCFraud will complement server-side techniques. According to the report published by WhiteOps [48], we notice that each year the amount of advertising fraud is increasing at a rapid rate. This clearly shows that the server-side techniques are inadequate to fight click-fraud. When malware control a large number of residential computers around the globe, it is very hard to detect low-noise attacks. Also, the use of VPNs, anonymized networks, etc. makes the detection tough from the server-side. FCFraud can make a positive change by curbing the number of bots in a botnet performing click-fraud.

DOI: 10.1002/smr

One of the major limitations of FCFraud is the inability to detect complex JavaScript and encrypted requests. These requests are almost impossible to detect at the operating system level without the help of the browser. One future direction of the research is to instrument the user processes so that they can communicate with the operating system and thus improve the detection rate drastically. We also plan to develop a method to periodically train the classifier on new dataset as ad URLs constantly change their structure to combat ad blockers. Lastly, our approach can be adopted to employ against other large-scale botnet attacks such as distributed denial of service (DDoS) and email spamming.

7. CONCLUSION

Nowadays, online advertising is a common form of business marketing and one of the reasons of the availability of free web contents or mobile apps. It is expanding rapidly in the advent of smart televisions and online content delivery services. As a result, fraudsters are also targeting the industry to drain money from the advertisers. One of the major threats for the online advertising industry is the click-fraud.

Sometimes a victim user's machine (desktop or mobile) unknowingly becomes a part of a larger click-fraud network by getting infected by a malware. In this paper, we develop a method, FCFraud, that protects innocent users by detecting the fraudulent processes that perform click-fraud silently. FCFraud can execute as a part of the operating system's anti-malware service. It inspects and analyzes web requests and input events from all user processes and applies RandomForest algorithm to automatically classify ad requests. After that, it detects fraudulent ad clicks using a number of heuristics. In our experimental evaluation, FCFraud successfully detects all the processes running in the background and performing click-fraud. It can block them from further accessing the network thus removing the machine from the botnet. We show that the overhead of FCFraud is reasonable in both desktop and mobile versions. We also build a cloud backend for the mobile version to provide users a choice to offload the analysis task to the cloud. Most major operating system vendors own or operate ad networks and advertising is one of their major sources of income. As a result, we believe that adding FCFraud to the operating system's anti-malware service can greatly serve the interests of the operating system vendors and online advertisers. It can also be a valuable addition to the server-side detection techniques used by the ad-networks to detect click-fraud.

ACKNOWLEDGEMENT

This work is partially supported by Mitacs Canada and Irdeto Canada.

REFERENCES

1. Internet ad spend to reach $121b in 2014, 23% of $537b total ad spend. http://techcrunch.com/2014/04/07/internet-ad-spend-to-reach-121b-in-2014-23-of-537b-total-ad-spend-ad-tech-gives-display-a-boost-over-search/. Accessed: 2015-02-18.
2. Mobile ad spending report. http://www.emarketer.com/Article/Mobile-Ad-Spend-Top-100-Billion-Worldwide-2016-51-of-Digital-Market/1012299. Accessed: 2016-02-09.
3. Wilbur KC, Zhu Y. Click fraud. *Journal of Marketing Science* 2009; **28**(2):293–308.
4. Zhang L, Guan Y. Detecting click fraud in pay-per-click streams of online advertising networks. *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2008; 77–84.
5. Juels A, Stamm S, Jakobsson M. Combating click fraud via premium clicks. *USENIX Security*, vol. 70, 2007.
6. Stone-Gross B, Stevens R, Zarras A, Kemmerer R, Kruegel C, Vigna G. Understanding fraudulent activities in online ad exchanges. *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*, ACM, 2011; 279–294.
7. Dave V, Guha S, Zhang Y. Measuring and fingerprinting click-spam in ad networks. *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM, 2012; 175–186.
8. Dave V, Guha S, Zhang Y. Viceroi: catching click-spam in search ad networks. *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*, ACM, 2013; 765–776.
9. Haddadi H. Fighting online click-fraud using bluff ads. *ACM SIGCOMM Computer Communication Review* 2010; **40**(2):21–25.

DOI: 10.1002/smr

10. Feily M, Shahrestani A, Ramadass S. A survey of botnet and botnet detection. *Proceedings of the 3rd International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, IEEE, 2009; 268–273.
11. Daswani N, Stoppelman M. The anatomy of clickbot. a. *Proceedings of the 1st Conference on Hot Topics in Understanding Botnets*, USENIX Association, 2007; 1–11.
12. Digital ad industry will gain $8.2 billion by eliminating fraud and flaws in internet. http://www.iab.com/news/digital-ad-industry-will-gain-8-2-billion-by-eliminating-fraud-and-flaws-in-internet-supply-chain-iab-ey-study-shows/. Accessed: 2016-02-06.
13. Iqbal MS, Zulkernine M, Jaafar F, Gu Y. Fcfraud: Fighting click-fraud from the user side. *Proceedings of the 16th IEEE International Symposium on High Assurance Systems Engineering (HASE 2016)*, IEEE, 2016; 157–164.
14. Facebook mobile ad revenue report. http://techcrunch.com/2016/01/27/facebook-earnings-q4-2015/. Accessed: 2016-02-09.
15. Kshetri N. The economics of click fraud. *IEEE Journal of Security & Privacy* 2010; **8**(3):45–53.
16. Miller B, Pearce P, Grier C, Kreibich C, Paxson V. What's clicking what? techniques and innovations of today's clickbots. *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2011; 164–183.
17. Gandhi M, Jakobsson M, Ratkiewicz J. Badvertisements: Stealthy click-fraud with unwitting accessories. *Journal of Digital Forensic Practice* 2006; **1**(2):131–142.
18. Jackson C, Barth A, Bortz A, Shao W, Boneh D. Protecting browsers from dns rebinding attacks. *ACM Transactions on the Web (TWEB)* 2009; **3**(1):2.
19. The spyware - click-fraud connection – and yahoo's role revisited. http://www.benedelman.org/news/040406-1.html#e1. Accessed: 2015-07-09.
20. Botnet implicated in click fraud scam. http://www.theregister.co.uk/2006/05/15/google_adword_scam/. Accessed: 2015-07-09.
21. Pandalabs uncovers a 'pay per click' botnet fraud. http://www.pandasecurity.com/about/press/viewnews.htm?noticia=7368&entorno=&ver=&pagina=&producto=. Accessed: 2015-07-09.
22. Zeroaccess click-fraud botnet. https://nakedsecurity.sophos.com/2015/01/31/zeroaccess-click-fraud-botnet-coughs-back-to-life/. Accessed: 2016-04-09.
23. Adwatcher: Advanced click fraud detection. http://www.adwatcher.com/click-fraud-features.php. Accessed: 2015-02-06.
24. Clickcease: Blocking click fraud. https://clickcease.com/. Accessed: 2015-02-06.
25. Clickreport: Click fruad detection and monitoring. http://clickreport.com/click-fraud-prevention. Accessed: 2015-02-06.
26. Metwally A, Agrawal D, Abbadi AE. Using association rules for fraud detection in web advertising networks. *Proceedings of the 31st International Conference on Very Large Databases*, VLDB Endowment, 2005; 169–180.
27. Metwally A, Agrawal D, El Abbadi A. Detectives: detecting coalition hit inflation attacks in advertising networks streams. *Proceedings of the 16th International Conference on World Wide Web*, ACM, 2007; 241–250.
28. Immorlica N, Jain K, Mahdian M, Talwar K. Click fraud resistant methods for learning click-through rates. *Proceedings of the Internet and Network Economics*. Springer, 2005; 34–45.
29. Xu H, Liu D, Koehl A, Wang H, Stavrou A. Click fraud detection on the advertiser side. *Proceedings of the 19th European Symposium on Research in Computer Security*. Springer, 2014; 419–438.
30. Pearce P, Felt AP, Nunez G, Wagner D. Addroid: Privilege separation for applications and advertisers in android. *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ACM, 2012; 71–72.
31. Shekhar S, Dietz M, Wallach DS. Adsplit: Separating smartphone advertising from applications. *Proceedings of the 21st USENIX Security Symposium*, 2012; 553–567.
32. Crussell J, Stevens R, Chen H. Madfraud: investigating ad fraud in android applications. *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ACM, 2014; 123–134.
33. Cho G, Cho J, Song Y, Choi D, Kim H. Combating online fraud attacks in mobile-based advertising. *EURASIP Journal on Information Security* 2016; **2016**(1):1–9.
34. The libpcap project. http://sourceforge.net/projects/libpcap/. Accessed: 2015-02-06.
35. Easylist ad block filter. https://easylist.adblockplus.org/en/. Accessed: 2015-02-06.
36. Purdy GN. *Linux iptables pocket reference*. O'Reilly Media, Inc., 2004.
37. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The weka data mining software: an update. *ACM SIGKDD explorations newsletter* 2009; **11**(1):10–18.
38. palmsource. Openbinder, version 1.0. http://www.angryredplanet.com/~hackbod/openbinder/docs/html/index.html Dec 28 2005. Accessed: 2017-08-05.
39. Quinlan JR. *C4. 5: Programs for machine learning*. Elsevier, 2014.
40. Breiman L. Random forests. *Journal of Machine Learning* 2001; **45**(1):5–32.
41. Phantomjs: Headless website browsing. http://phantomjs.org/. Accessed: 2015-01-09.
42. Selenium: Browser automation. http://www.seleniumhq.org/. Accessed: 2015-01-09.
43. Sissel J. Xdotool-fake keyboard/mouse input, window management, and more 2013.
44. Odroid xu3. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127. Accessed: 2015-07-09.
45. Hardkernel. http://www.hardkernel.com/main/main.php. Accessed: 2016-02-03.
46. Jacobson V, Leres C, McCanne S. Tcpdump public repository. *Web page at http://www. tcpdump. org* 2003; .
47. Odroid smart power. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137361754360. Accessed: 2015-07-09.
48. Bot fraud trend. http://www.whiteops.com/downloads/the-bot-baseline-fraud-in-digital-advertising. Accessed: 2016-10-06.

DOI: 10.1002/smr