FCFraud: Fighting Click-Fraud from the User Side

Md Shahrear Iqbal*, Mohammad Zulkernine*, Fehmi Jaafar*, Yuan Gu†
*Queen's University, Kingston, Ontario, Canada
{iqbal,mzulker,jaafar}@cs.queensu.ca
†Irdeto Canada, Kanata, Ontario, Canada
yuan.gu@irdeto.com

***Abstract*—Pay-Per-Click (PPC) ad networks charge advertisers for every click on their ads. Click-fraud happens when a user or an automated software clicks on an ad with a malicious intent and advertisers need to pay for those valueless clicks. Click-fraud has been proved to be a serious problem for the online advertisement industry. Although it has attracted much attention from the security community, the direct victims of click-fraud, the advertisers, still lack confidence in the click-fraud detection techniques. Among many forms of click-fraud, botnets with the automated clickers are the most severe ones. In this paper, we present a technique for detecting automated clickers from the user side. Normal internet users are victimized by malicious attackers (e.g., bot-master of a botnet) and the attackers infect and use their machines to defraud advertisers. We propose a technique to Fight Click-Fraud, FCFraud, which can be integrated into the operating system. Since most modern operating systems already provide some kind of anti-malware service, our proposed technique can be implemented with a negligible overhead. 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% accurate in classifying ad requests from all user processes and it is 100% successful in finding the fraudulent processes.**

***Keywords*—Online advertising, click-fraud, malware detection.**

I. Introduction

Online advertising is a form of marketing which uses websites to deliver promotional messages to consumers. It includes email marketing, search engine marketing (SEM), social media marketing, many types of display advertising, and mobile advertising. It is the main financial incentive for free web contents and services as well as free mobile apps. The online advertising industry has expanded rapidly and grown into a $121 billion industry with revenue projected to reach $161 billion by 2016 [1]. The largest revenue shares within internet advertising are generated by display-based and search-based advertising.

Advertisers and 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. Click-fraud is the practice of deceptively clicking on online ads with the intention of either increasing third party website revenues or exhausting an advertiser's budget.

Ad networks mostly use server-based techniques to detect advertising frauds as they do not have enough control over the client machines. They gather information from different sources about the user, the machine and the behavior of the user. Then, they apply machine learning or pattern recognition techniques to identify suspicious clicks.

The most severe attacks are the attacks by the botnets which are used extensively in recent years to launch large-scale attacks. A botnet is a network of malware-infected machines that are controlled by a bot-master. The users of such machines are normally unaware of the fact that their machines are being compromised and used by the attackers. FCFraud particularly protects this group of users from being exploited.

This paper proposes a technique, FCFraud, which can be incorporated as a part of an operating system's anti-malware service. FCFraud inspects the HTTP packets from all the user processes along with the real events from the hardware mouse devices. It detects the fraudulent processes in user machines that programmatically click on ads silently while executing in the background. It is impossible for the fraudulent processes to generate real mouse clicks. These processes either make a new HTTP request to simulate a click on an ad or generate software simulated clicks which are distinguishable from real mouse clicks. In summary, in this paper, we make the following contributions:

- We take the first step to propose a click-fraud prevention technique, FCFraud, on the user side which can be a valuable addition to the server-side detection techniques.
- We propose to add FCFraud as a part of an operating system's anti-malware service.
- We experimented using 25 popular websites (a total of 7,708 HTTP requests) to examine the effectiveness of FCFraud.

The remainder of the paper is organized as follows. Section II describes click-fraud and details about automated clickers. Section III relates our study with the previous work. Section IV presents our click-fraud prevention technique and Section V describes the experimental evaluation results. Section VI discusses FCFraud's effectiveness, limitations and a number of future directions. Finally, we conclude in Section VII.

II. Background

In this section, we first explain how the online advertising works. Then, we present a brief discussion on the click-fraud along with an example of the botnet click-fraud.

A. Click-fraud

Click-fraud occurs when illegitimate sources click on online ads with a malicious intent. Such clicks are often called "invalid clicks". Invalid clicks are any clicks that an ad network chooses not to charge for. When clicks are marked invalid,

the user agent that issued the click is still directed to an advertiser's website. Since the intent is only inside the mind of the person issuing the click or inside the mind of the author of the software that issues clicks, it is difficult to know with certainty whether a click is fraudulent. Detecting all invalid clicks is not trivial due to the server-based detection techniques used by the ad networks. In fact, about 36% of all web traffic is considered fake according to the estimates cited by the Interactive Advertising Bureau trade group [2]. A study of the digital security firm White Ops and the Association of National Advertisers, estimates that advertisers will lose $6.3 billion in 2015 due to click-frauds [3].

B. Automated clickers

Clickbots are special software that can click on online ads automatically. Today's 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 users or bot-masters. 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 bot-master. A bot-master can leverage millions of clickbots to perform automatic and large-scale click-fraud attacks. The following description illustrates how a victim host conducts click-fraud under the command of a bot-master. First, the bot-master uses internet 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 bot-master. 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 [4]. 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.

III. Related Work

Over the last decade, researchers from big companies like Google and Yahoo are investigating the problem of click-fraud [5], [6]. Since advertising revenue is one of the major sources of income of these companies and many independent publishers, click-fraud became a major threat to the survival of free contents on the web [7]. Advertisers also have a strong interest to combat all sorts of advertising frauds that are draining their money. In this section, we will discuss the most pertinent literature about click-fraud detection.

Metwally et al. [8] 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. Immorlica et al. [9] studied fraudulent clicks and presented a click-fraud resistant method for learning the click through rate of advertisements. Haddadi [10] suggested that advertisers can use bluff ads to detect fraudulent clicks on their ads. While bluff ads may be effective in detecting click-fraud, advertisers have to spend extra money on those bluff ads. Also, for the publishers, it is not good to show meaningless ads to the real users. Dave et al. [11] presented an approach for catching click spam from an ad network's perspective. It is designed based on an 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.

Schulte et al. [12] detected malware using program interactive challenge (PIC) mechanism. However, in their approach, an intermediate proxy has to be introduced to examine all the HTTP traffic between a client and a server. Similarly, Jang et al. [13] proposed a framework to match user intents with the network traffic. Using the framework, they tried to distinguish between human and malware generated traffic. They also used a proxy machine to monitor network traffic. This is technically difficult to implement in real environments and we address the problem by implementing our approach as a part of the operating system. Xu et al. [14] 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.

The most closely related work to FCFraud is that of Crussell et al. [15]. They 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. They created HTTP request trees from the collected packets and used machine learning to automatically identify ad requests. They achieved an overall class-weighted accuracy of 85.9% in classifying ad requests from all the captured network traffic. Finally, they reported a number of ad-related frauds in Android apps. In our work, we use the concept of HTTP request trees for detecting ad requests. However, our proposed technique, FCFraud, executes in the user operating systems to prevent click-fraud.

To summarize, in the field of click-fraud, most research focuses on the detection from the server-side. Publishers, ad networks or the advertisers use global advertising web traffic and their own proprietary technology to detect fraudulent clicks [16]–[18]. In our understanding, click-fraud is a user side behavior. Therefore, our approach focuses to detect click-fraud on the user side.

IV. The Click-fraud Prevention Technique

This section describes the steps of FCFraud. To prevent click-fraud, FCFraud tries to detect programs that run in the background, implement browser functionalities and perform click-fraud stealthily.

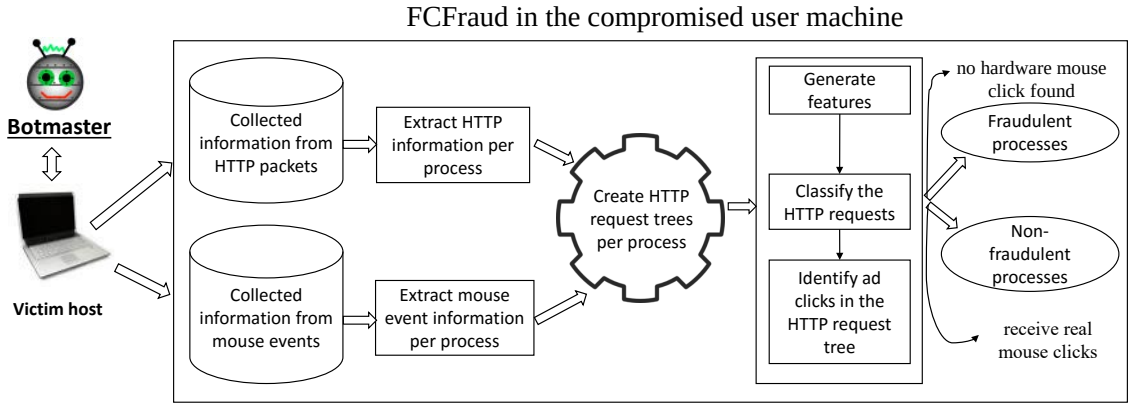

Fig. 1: A user side approach for preventing click-fraud.

As shown in Figure 1, FCFraud first extracts HTTP packet and mouse event information per process and 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. We mark a process fraudulent if it does not generate real mouse clicks, however, its tree contains ad requests and clicks. Processes that interact with a real user and receive hardware mouse clicks are marked non-fraudulent. In a nutshell, the steps are shown in Figure 2. We describe each step of the technique in the following subsections.

```
while Machine is ON do
   -Capture HTTP Packets
   -Capture input events
   -Apply easylist ad-block filter on 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
      -Identify ad clicks in the HTTP request trees
   end for
   -Detect and block the fraudulent processes
end while
```

Fig. 2: Click-fraud prevention algorithm

A. Collect information from the HTTP requests

In Figure 3, we demonstrate how FCFraud collects information from each running process. FCFraud uses the libpcap [19] packet capture library to capture HTTP packets from the network interface. At the time of capturing, it looks into the "/proc" file system to find out the process which is currently using the source port.

For each process, FCFraud records different fields from the HTTP request and response. 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 request URI (uniform resource identifier). From the response, it records location, content type, content length, content encoding, HTTP version, status code, and reason phrase. FCFraud saves all these information in an SQLite database for later steps. In this step, it neither inspects the body of the requests nor it saves the packets as it may require gigabytes of storage space.

EasyList ad blocking filter. EasyList [20] is a popular ad block filter maintained by the online community and used in ad blocking add-ons of web browsers. It can filter ad-related contents via URL filters, DOM element filters, and third-party advertisement domain filters. FCFraud uses a version of EasyList that contains a total of 45,423 rules. After saving HTTP packet information, FCFraud checks each of the URL against all these rules and saves the decision in the database. This boolean information is used as a primary filter to detect processes with ad-related internet traffic. It is also used as a feature in the ad request classifier later on. Figure 3 presents the steps described in Section IV-A.

B. Record real mouse events

FCFraud records the timestamp of each mouse event received by a process. It records events from hardware mouse devices only (listens inputs from hardware ports). To accomplish this, it executes separate user-level threads for each mouse device present in the system while capturing HTTP packets. At the time of recording, it determines the top window id of the x-window system and find all the process ids 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 mouse clicks.

C. Create the HTTP request trees

HTTP requests of the processes can be grouped logically together. 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 referrer 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. If we can construct a tree that represents the exact scenario of the

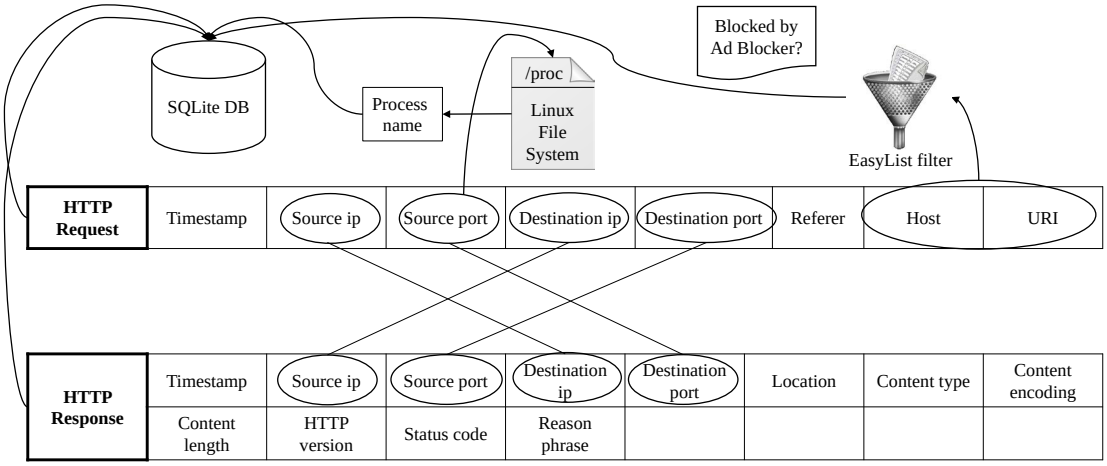

Fig. 3: Capturing HTTP packets and extracting information per process.

browsing, then it will be easier to analyze and automatically detect ad clicks.

We represent each HTTP request and its corresponding response as a single node in the request tree and connect two nodes if:

1) The latter node contains the request referer field set to the host of the former node. 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 4. The edges are marked as "REFERER" and "LOCATION" based on the above-described rules.

D. Generate features and classify the HTTP requests

To detect ad clicks, 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 the EasyList filter as described in Section IV-A. We use WEKA machine learning library [21] to classify HTTP requests.

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

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 is a summary of the key feature categories we use to train the classifier.

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

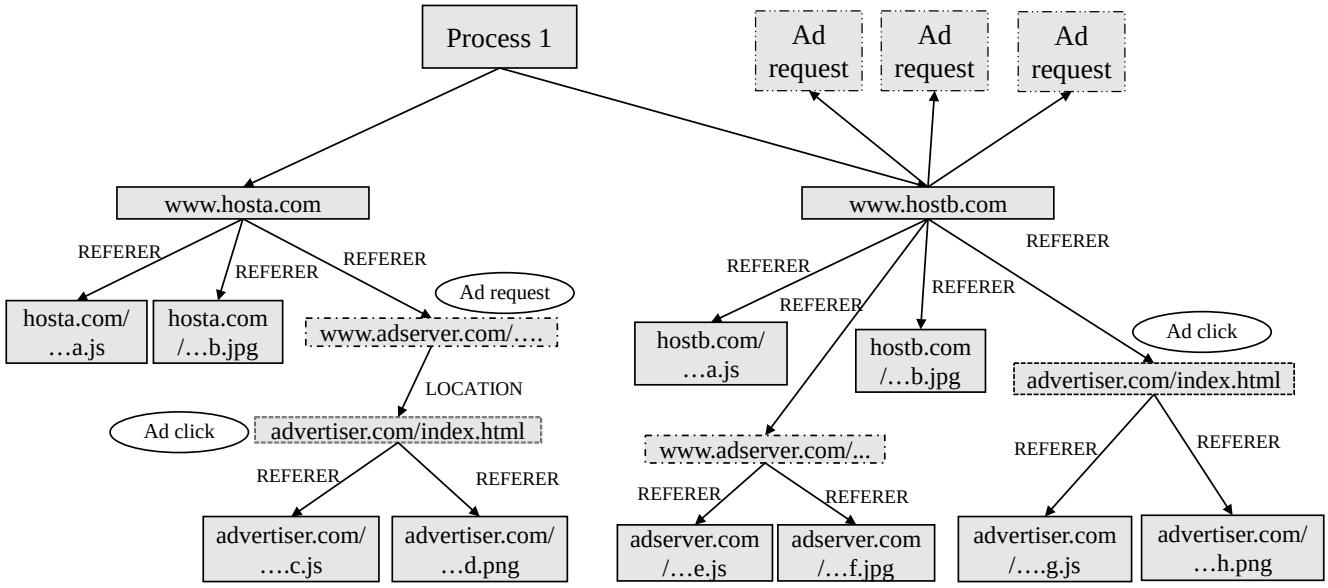

Fig. 4: An example of HTTP request trees of a process. We encircle the ad click and ad request nodes.

send many information about the particular client machine and the browser 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. From this set, we select features like the height of the subtree rooted at each node, the number of blocked URLs in the subtree, the number of images and JavaScript requests in the subtree, the number of different domains in the subtree, etc.

2) Metrics for evaluating the classifiers

We use five classification algorithms to classify our data, namely, Naive Bayes, Support Vector Machines (SVM), K-Nearest Neighbors, C4.5 [22], and Random Forest [23]. We use WEKA's default configuration for the five classifiers. We will compare their performance in Section V. Here, for all the classifiers, the output is either "ad" or "notad" for every input instance. To evaluate the average classification accuracy of each classifier, we use the k-fold cross validation (CV) with k=3. By using the k-fold CV, we ensure 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. The formulas for calculating the precision and the false positive rate are given below:

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

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

In our case, we like to minimize the false positive rate and maximize the accuracy of the "ad" class.

E. Identify ad clicks in the HTTP request trees

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. 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 likely 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 clicks 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 4. 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

Classification Algorithm	Avg. Accuracy (%)	Precision (%)	FP Rate (%)
NaiveBayes	92.46	59.05	7.48
SVM	96.85	100	0
C4.5	99.27	95.09	0.59
2kNN	99.27	95.64	0.52
5kNN	99.29	95.64	0.52
RandomForest	99.57	97.43	0.30

TABLE I: Performance of different machine learning classifiers to classify ad requests over 3-fold cross-validation.

are made to static resources, such as CSS files which can be removed from the list without hampering the performance of the ad click detection.

F. Detect and block the fraudulent processes

To detect ad fraud, FCFraud analyzes the HTTP request trees and the collected mouse events. If a process clicks on an ad using a software simulated click or an independent HTTP request (without a mouse click) while executing in the background, then FCFraud marks the process as fraudulent. It uses the mouse event database to determine whether this particular process generated any real mouse clicks (i.e., a real user interacts with it). If FCFraud identifies any fraudulent process, it notifies the user and blocks the process from accessing the network.

V. Experimental Evaluation

In this section, we first describe how we execute FCFraud in our setup. Then, we create a ground truth dataset by visiting a number of websites. We use 3-fold cross-validation to compare the performance of the five classifiers. Here, we choose the best classifier to be used in FCFraud. After that, we execute FCFraud to detect fraudulent processes in the machine. Finally, we observe the effectiveness of FCFraud based on how many clickbots it detects.

A. Execution of FCFraud

In our experiment, FCFraud executes as a daemon service with root privileges. It inspects HTTP requests and saves necessary information along with the mouse events in a database. Then, FCFraud analyzes the requests and alarms the user periodically. In our case, it took 229.336 seconds to analyze the inspected HTTP requests (around 4.3MB of captured information) in a Pentium Core i7-4770 3.4GHz machine with 16GB of RAM and Ubuntu 14.04 64-bit operating system. Notably, the database contains only textual information and the operating system can easily send it to the cloud for the analysis if the user machine is not powerful.

B. Visited websites

We select 25 websites containing advertisements for our experiment. We try to pick the most popular categories (from `www.alexa.com`) of websites such as articles, auto, weblog, computer games, science, technology, travel, and weather.

C. Click bots

We use three kinds of bots to generate the fraudulent traffic. Two of them use browser drivers to control Google chrome and PhantomJS [24]. 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 Selenium WebDriver [25]. 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. PhantomJS is a headless browser and does not create any human viewable window. The third bot is an independent program, which generates HTTP traffic and parses the response. We use a web browser control to implement the feature. We execute the bots both sequentially and concurrently to examine the effectiveness of FCFraud.

In the three bots, we supply website addresses and XPaths of the web elements containing the ads. XPath is a query language for selecting nodes in an XML document. The bots first visit the websites given to them and generate clicks on the web elements containing the ads which redirect the browser to the advertisers' pages. Occasionally, the bots generate multiple clicks on the same page if multiple XPaths are supplied to them.

D. Ground truth for the ad request classifiers

To build the ground truth dataset for identifying ad requests, we visit all the websites using the Firefox web browser and inspect the ad elements by using the Firebug extension. We manually classify 809 ad requests from the 7,708 HTTP requests generated by the clickbots and the Firefox browser. We do not classify the tracking and conversion requests to the ad networks and advertisers as they are not requests for serving ads. The total number of ad-related domains in the dataset is 13.

E. Identifying the ad requests

After we build the training dataset, we use the selected classifiers to classify the HTTP requests. The classifiers classify a request as either "ad" or "notad". We consider the instances from the class "ad" as positive.

Table I shows the comparison of the average accuracy, the precision and the false positive rate of the various classifiers. In our experiment, RandomForest has the highest average accuracy and precision of 99.57% and 97.43%. It also has the lowest false positive rate of 0.30%. Though the average accuracy, the precision and the false positive rate of the SVM algorithm is promising, it has a poor recall value of only 69.96% as shown in Figure 5. RandomForest produces the

highest recall value of 98.51%. 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.

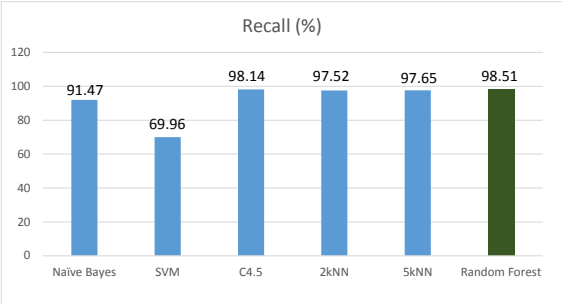

Fig. 5: Accuracy of the positive class (recall) of different classifiers.

Table II presents the confusion matrix of the RandomForest classifier. We discussed in the related work section that this step of FCFraud uses the concept of HTTP request trees from [15]. Nevertheless, we are unable to quantifiably compare our classification results with their results. Their dataset is for Android apps and ad request formats in mobile platforms are different. Also, we have a number of different heuristics to build the HTTP request trees so that they resemble more to the desktop browsing environment.

	NOTAD	AD	Recall
NOTAD	6878	21	99.7%
AD	12	797	98.5%
Precision	99.8%	97.4%	

TABLE II: Confusion matrix of our RandomForest classifier, computed using 3-fold cross validation. The accuracy is 99.6%.

F. Detecting ad clicks in request trees

Detecting ad clicks is the trickiest part of our technique. In Figure 6, we report the results of the detection. It detects 24 ad clicks among a total of 28. It could not detect 4 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. Here, FCFraud reports a total of 19 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 the precision of the ad click detection are 85.71% and 55.8%.

G. Detecting and blocking fraudulent processes

FCFraud successfully detects all the three clickbots as fraudulent. We find that all the ad clicks are detected as fraudulent in the request trees generated by the clickbots. After the detection, FCFraud notifies about the incident and blocks the fraudulent processes from accessing the network. Without network access, the processes become ineffective for the attack as they cannot communicate with the bot-master. Also, it does not make any difference whether we run the bots concurrently or sequentially. In both the cases, FCFraud performs equally.

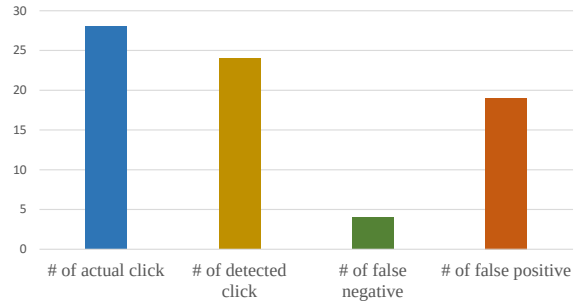

Fig. 6: Evaluation result for the detection of ad clicks.

VI. Discussion

In this section, we discuss various issues of the click-fraud problem, our approach, our limitations, and probable future directions.

A. Effectiveness of FCFraud against click-fraud

We primarily design FCFraud to thwart processes which are part of a large click-fraud network. These processes perform several types of advertising frauds. For example, some of them manipulate search engine results by constantly searching bot-master provided keywords and click on results. Some of them browse affiliate web sites and click on advertisements. FCFraud successfully blocks all those processes and thus shrinking the size of the botnet.

Another issue is the timing of the click-fraud attack. Many of the malware tries to generate fraudulent clicks when the computer is being actively used by a user. It is not an issue for FCFruad. It will detect the malware whether it executes the attack in a busy time period or not. Some malware create multiple processes and frequently change their executables. In both the cases, FCFraud can block them as it traces the file path of the parent process that spawned all those children. FCFraud blocks the process using its physical file path. As a result, it will block the malware even if the executable changes.

FCFraud will also work in case a user has a touchscreen monitor. Touch screens generate events similar to the mouse events discussed in the paper and can be captured using the same technique.

B. Applicability of FCFraud in mobile devices

Since mobile advertising is on the rise, it is high time to enhance the mobile operating systems with anti-malware capabilities. We are now actively working to port our solution to the mobile environment. We are experimenting with the Android operating system that uses a modified Linux kernel. We find that capturing touch events and HTTP requests require a little modification. Another advantage of the mobile devices

is that they allow only one process to be in the foreground, which eases the task of associating touch events with processes. We are confident that FCFraud will be equally effective in the mobile operating systems. Also, as mentioned before, the analysis can be done in the cloud to save mobile devices' battery life.

C. Limitations and Future work

One of the major limitations in FCFraud is the inability to detect complex JavaScript and encrypted requests which is 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 improving the detection rate drastically.

Another limitation is the number of false-positives in the ad click detection process. Nowadays, ad URLs are constantly changing their structure to combat ad blockers. That is why we are developing a method to periodically train the classifier on new data.

Lastly, our approach can be effective against other large-scale botnet attacks such as distributed denial of service (DDoS) and email spamming.

VII. Conclusion

Nowadays, online advertising is a common form of business marketing and one of the reasons of the availability of free web contents. 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 unknowingly becomes a part of a larger click-fraud network by downloading or getting infected by a malware. In this paper, we develop a technique, FCFraud, that protects innocent users by detecting the fraudulent processes that perform click-fraud silently. FCFraud executes as a part of the operating system's anti-malware service. It inspects and analyzes web requests and mouse events from all the user processes and applies RandomForest algorithm to automatically classify the 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 blocks them from further accessing the network thus removing the machine from the botnet. 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 and it can be a valuable addition to the server-based detection techniques.

Acknowledgment

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] "A 'crisis' in online ads: One-third of traffic is bogus," http://www.wsj.com/articles/SB10001424052702304026304579453253860786362, accessed: 2015-01-09.

[3] "Fraud from bots represents a loss of $6 billion in digital advertising," http://www.reuters.com/article/2014/12/09/us-advertising-fraud-study-idUSKBN0JN0AW20141209, accessed: 2015-02-06.

[4] N. Daswani and M. Stoppelman, "The anatomy of clickbot. a," in *Proceedings of the 1st Conference on Hot Topics in Understanding Botnets*. USENIX Association, 2007, pp. 11–11.

[5] "Why google and publishers care about invalid traffic," http://www.google.ca/ads/adtrafficquality/, accessed: 2015-02-06.

[6] "How does yahoo protect me from click fraud?" http://help.yahoo.com/l/in/yahoo/ysm/sps/faqs/click_fraud.html, accessed: 2015-02-06.

[7] N. Kshetri, "The economics of click fraud," *IEEE Journal of Security & Privacy*, vol. 8, no. 3, pp. 45–53, 2010.

[8] A. Metwally, D. Agrawal, and A. E. Abbadi, "Using association rules for fraud detection in web advertising networks," in *Proceedings of the 31st International Conference on Very Large Databases*. VLDB Endowment, 2005, pp. 169–180.

[9] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar, "Click fraud resistant methods for learning click-through rates," in *Proceedings of the Internet and Network Economics*. Springer, 2005, pp. 34–45.

[10] H. Haddadi, "Fighting online click-fraud using bluff ads," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 21–25, 2010.

[11] V. Dave, S. Guha, and Y. Zhang, "Viceroi: catching click-spam in search ad networks," in *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013, pp. 765–776.

[12] B. Schulte, H. Andrianakis, K. Sun, and A. Stavrou, "Netgator: malware detection using program interactive challenges," in *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 164–183.

[13] Y. Jang, S. P. Chung, B. D. Payne, and W. Lee, "Gyrus: A framework for user-intent monitoring of text-based networked applications," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 79–93.

[14] H. Xu, D. Liu, A. Koehl, H. Wang, and A. Stavrou, "Click fraud detection on the advertiser side," in *Proceedings of the 19th European Symposium on Research in Computer Security*. Springer, 2014, pp. 419–438.

[15] J. Crussell, R. Stevens, and H. Chen, "Madfraud: investigating ad fraud in android applications," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2014, pp. 123–134.

[16] "Adwatcher: Advanced click fraud detection," http://www.adwatcher.com/click-fraud-features.php, accessed: 2015-02-06.

[17] "Clickcease: Blocking click fraud," https://clickcease.com/, accessed: 2015-02-06.

[18] "Clickreport: Click fruad detection and monitoring," http://clickreport.com/click-fraud-prevention, accessed: 2015-02-06.

[19] "The libpcap project," http://sourceforge.net/projects/libpcap/, accessed: 2015-02-06.

[20] "Easylist ad block filter," https://easylist.adblockplus.org/en/, accessed: 2015-02-06.

[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[22] J. R. Quinlan, *C4. 5: Programs for machine learning*. Elsevier, 2014.

[23] L. Breiman, "Random forests," *Journal of Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[24] "Phantomjs: Headless website browsing," http://phantomjs.org/, accessed: 2015-01-09.

[25] "Selenium: Browser automation," http://www.seleniumhq.org/, accessed: 2015-01-09.