Click Fraud Detection and Prevention System for Ad Networks

Paulo S. Almeida, João J. C. Gondim

***Abstract*—Click fraud detection consists of identifying the intention behind received clicks, given only technical data and context information. Reviewing concepts involved in click fraud practices and related work, a system that detects and prevents this type of fraud is proposed and implemented. The system is based and implemented on an ad network, one of the 3 main agents in the online ad environment, and for its validation, 3 servers were used, representing the publisher, the ad network with the system implemented and the announcer, and a bot that attempts to commit a click fraud.**

***Index Terms*—Click fraud, online security, bots, system architecture**

I. Introduction

THE publicity domain grows continually by the day. One of the reports from eMarketer [1] states an estimated growth of nearly 16% for the digital advertisement area in the United States compared to the previous year, summing almost 83 million dollars. That's equivalent to roughly 40% of the overall investment in advertising, estimated as 205.06 billion dollars. With these numbers, it is possible to see the importance and interest gained by the area recently, which includes related technical aspects, such as click fraud detection.

A. Motivation

Click fraud detection is an inherently nebulous field. In broad terms, it consists of identifying the intention behind the received clicks, given only technical data (such as the IP address and other information provided by HTTP requests) and contextual information (previous accesses from the same IP, for example). Thus, malicious click detection involves comparing every access behavior with what's expected from normal users, but, that is difficult to formalize and context dependent behavior is nondeterministic and context dependent.

Moreover, current literature is lacking in certain aspects. Many studies focus on the advertiser's side, such as [2], and few on how to apply click fraud detection techniques on ad networks, the middle agent between the publisher and advertiser, all of which are parties involved in the online publicity area (see Section II); and relatively shallow elaboration on the technical details and inner workings of established click fraud detection systems, giving little insight on the design decisions and compromises that went on behind the creation of such applications.

This work was supported in part by the Institutional Security Office of the Presidency of the Republic of Brazil (GSI/PR) Grant 002/2017.

Paulo S. Almeida was with the Computer Science Department, Universidade de Brasília, Brasília, Distrito Federal. He is now with Laboratório LATITUDE, Universidade de Brasília (e-mail: paulo.almeida@redes.unb.br).

João J. C. Gondim is with the Computer Science Department, Universidade de Brasília, Brasília, Distrito Federal (e-mail: gondim@unb.br).

B. Objective

The objective is to propose a system of click fraud detection and prevention applied to an ad network. Ad network's interest in preventing frauds lies in their relationship with advertisers: the latter wishes for the best click quality possible (or, the biggest number of interested users over the least of resources used possible), and directly pays the ad network for such traffic. Bad clicks directly hurt the advertisers' goals, and thus their wish to interact with networks that may bring such clicks. The approach is to provide is to provide an introduction into how one might implement a system as described above, but not necessarily for just an agent in the area, going into details such as design choices and architecture.

C. Outline

Throughout this report the path taken to create the defense system will be elaborated in detail. In Section II - Technical Review, is an overview of basic concepts necessary for a good understanding of the rest of the report, including the online advertisement agents and previous studies on similar subjects. Section III - Problem Description goes over the problems that we aim to address throughout the study and development process like usual click fraud types. The theory and concepts brought up and expanded upon, or proposed by this study, like the various rules and their classifications, to create the system's theoretical basis, are in Section IV - Proposal. Continuing that, Section V - Implementation talks about the implementation details that had to be considered on the process of programming the system's theory, including creating an attacking bot to test the system. Those are present in Section VI - Tests and Results, which shows and explains the results obtained. Section VII - Conclusion serves as a closing off for the study and compares the results with expectations, along with suggestions for system improvement.

II. Technical Review

TO be able to elaborate on the process of creation and design of the system, a solid understanding of the online publicity field's inner workings is required, along with a glance at what studies have already been published on the area and identify lacking parts in order to propose new system that can be relevant.

A. Online Advertisement Concepts

Online advertisement involves primarily four agents [3] [4]:

- The **advertiser** or **announcer** wishes to publicize their product or service to a target public that may be interested in consuming what the announcer has to offer.
- A **publisher** is someone who has their own publicity platform, such as a site or a blog, and is able to show the advertiser's product to the visitors of said platform. Often the content in the publisher's platform and that of the advertiser's product are of similar nature, though not always.
- **Users** are any visitors to the publisher's website, who may be interested and click on the advertiser's ad, perhaps even buying their product or otherwise doing a directly financially relevant action for the advertiser, thanks to the publisher's publicity.
- At last, the **ad network** is a middleman between the advertisers and publishers. Their objective is to connect advertisers interested in the publicity with publishers willing to offer such service. Some ad networks take a dynamic approach by deciding which ad to show for a given user accessing a given publisher's site.

Part of the growing interest in the area comes from the profit it may generate for any of the parts. The most common scenario is for the advertiser to pay for the services of either the publisher or the ad network (which in turn pays the involved publishers). How exactly this payment is measured varies, and different advertisers are interested in different types of audience and interactions with their ads. Usual payment methods are:

- Pay-Per-Impression (PPI), in which the advertiser is interested in how many users simply visualize their ad. Impressions are for the most part trivial to obtain, and hardly correspond with any gain for the advertiser by themselves, so the payment received per impression is extremely low and this method to be less common.
- Pay-Per-Mille (PPM) is a more utilized variant of PPI, where the payment is given based on how many thousands of views the ads obtained, making this method a more realistic payment option.
- Pay-Per-Click (PPC) cares about the quantity of clicks the ads have received. Often payment is based on the click-through rate that the publisher or ad network expected to get for this specific ad. This method is the focus of this study.
- A stricter method is Pay-Per-Conversion or Pay-Per-Action (PPA), where payment only happens based on the number of users which actually perform a certain action within the advertiser's domain, such as buying a product or signing up on their site.

An example of a PCC method and how it works is in Figure 1. The user visits the publisher's site (1), and then asks for the ad data that's provided by the ad network to the publisher's site (2). The network will proceed by choosing the ad they believe the user is more likely to be interested in (3,4) and sends the data to the user (5). If the user is indeed interested by the ad, they will click it, be redirected to the ad network's domain (6), and then to the advertiser's site (7). After this process, the announcer will eventually pay the ad network for redirecting a user to their site (8), and the ad network will in turn also pay the publisher for reaching to the user (9).

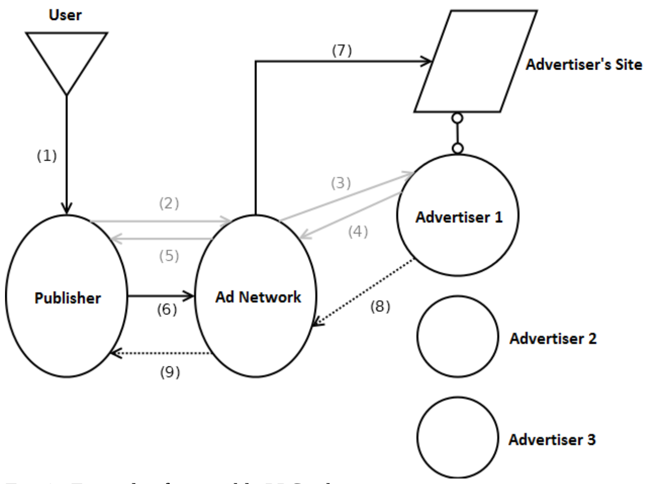


Fig. 1. Example of a possible PPC scheme.

Considering PPC the payment method, **click fraud** is the attack of interest. A click fraud happens when a malicious agent executes an illegitimate click on an ad, which is a click that comes from a user that doesn't have real interest in what's being advertised. Clicks may be generated manually (by hiring people to click repeatedly on certain ads), or, more often, automated (using bots).

B. HTTP Protocol

Understanding the nuances of the HTTP protocol is also important for understanding how the developed system works. HTTP stands for *HyperText Transfer Protocol* and is the base upon which a big part of internet applications are built. The many reports from the RFC 7230 family [5] [6] are the official source for the protocol's inner workings and other details. The Hypertext Transfer Protocol (HTTP) is a stateless application-level request/response protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems. One of the key aspects of this definition is HTTP’s statelessness, which is to say, there should be no persistence of data between one HTTP communication and the next, even if they're between the same parties.

Communication carried through HTTP takes form of someone sending an HTTP Request, which must be answered with an HTTP Response from the request's target. HTTP Request packets are essentially the only input to identify malicious click, apart from click context. They provide essential information such as the IP address and headers from their sender.

Cookies, although not part of HTTP, are another important component in web applications. They are a way of retaining state on communications with the HTTP Request's sender, and consist of a key and value pair, both of which can be any string the Request's receiver wishes. The presence of certain cookies in new HTTP Requests can be verified, allowing the receiver to take different options depending on the cookie's presence or values. Their use in the system will be clarified in Sections IV and V.

C. Related Work

Kitts et al. [7] present an overview on the *Microsoft adCenter* system and some of the design aspects of it. Although precise details aren't provided, a good idea of the overall workings of a defense system applied to an agent that's both publisher and ad network is given. Xu et al. [2] provide a detailed report on the techniques used for a fraud detection system on the advertiser side, which are key to a better understanding on how one would go about creating similar applications and general detection techniques. However, it is evident the lack of reports on systems that are used by ad networks, and a lack of elaboration on the more technical details and implementations of the tools used to detect and prevent click frauds.

On click fraud attack methods, [8] is a report about the inner workings of a click fraud *botnet*, and presents the concept of a low-frequency attack, a click fraud attempt that occurs over a long period of time, using a small number of clicks over short time slots, like 3 fake clicks per day, as to not be detected by methods that look at expected click rates for a given ad. Other attack reports include [9], [10] and [11], all of which go over complex fraud cases that happened throughout the years. Both concepts of botnets and low-frequency attacks are explained in section III.

III. Problem Description

GIVEN the scarce published material that focuses on click fraud detection systems on ad networks, the main focus will be on the development of a system for that purpose. A *dual approach* is adopted: the idea is to consider an attack and build the system around defending against this attack. Attacks start simple and evolve in complexity, and so does system's defense accordingly. This process goes on incrementally until blocking off what's believed to be the most usual forms of attacks.

Frauds, as already mentioned, can be done either manually or with the use of automatized bots.

Click frauds committed by people, at their most basic, would usually consist of the attacker accessing the publisher's page, and from there clicking the ad or ads that they are meant to. Improvements on this method of attack usually involve scaling it by involving many hired attackers, all with different computers, often within the same room or building, to raise click throughput. Attackers could also be instructed to browse the advertiser's page briefly, to better imitate a legitimately interested client.

Looking at how a bot attacks, on the other hand, first consider that the most basic way of registering a click is as an access to a specific ad URL, and thus the most basic type of fraud would be to simply go to the URL with a single HTTP request. More complex forms of attack would make use of patterns found in normal user behavior, namely expected fields in the HTTP packets, accessing all the expected URLs through the process, such as images, and not having a consistent time difference between different accesses.

More sophisticated types of fraud use **botnets**. They are networks of infected machines, previously exposed to some type of malware meant to give some form of access to victims' computers. Those machines receive instructions on how to operate from a control server or from another infected machine, which in turn may also receive its own instructions from the main server or another module, and so on [8]. Attacks themselves are often conducted without knowledge of the infected machine's owner, running in the background. Bot malware may use the user's own behavior for fraud, such as redirecting them from sites they intended to visit to sites the attacker wishes to grant clicks. This effectively gives the attack some degree of human characteristics, namely the machine's owner's access behavior and a legitimate HTTP packet, making this type of attack is difficult to identify. An illustration of the attack can be seen in figure 2.

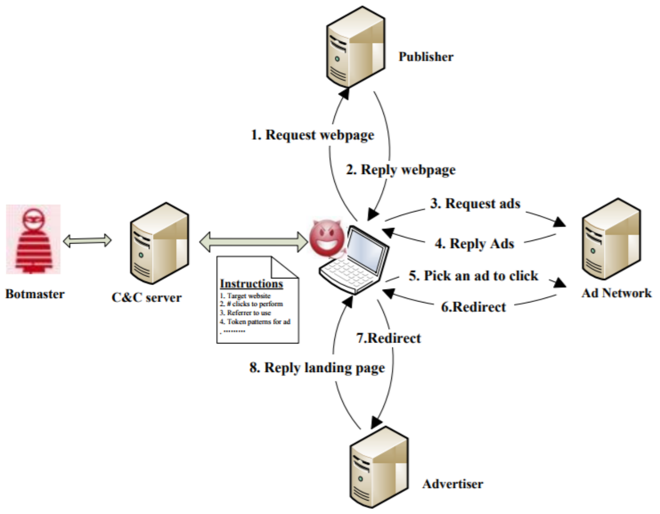


Fig. 2. Botnet attack with a compromised machine. Source: [12]

Other complex kind of fraud is the **low-frequency attack**. One common detection technique is to analyze expected click-through rate, a prediction on how many clicks an ad will receive over a period, mainly during certain times of the day and week, which is arrived at using advanced statistical analysis [12] [13]. That technique would then be able to notice unusual increases in activity for the ad it is analyzing. As attackers usually have more to gain with a larger number of clicks, this type of defense is very strong and will catch a good amount of simpler attacks.

The low-frequency attack consists, then, of countering this defense by only interacting with the ad very few times over certain periods. Considerable time between each access makes it difficult to detect frauds as they are happening, even by professionals, and those attacks are often only detected months after they've already started [8] [9]. The combination of botnets and low-frequency attacks can be a challenge to defense systems, since it might split the already small quantity of accesses between different IP addresses with little trouble, turning detection even more difficult.

IV. Proposal

THE overall architecture of the system is based on the proposal of [7]. Terms such as "rules" will appear in both works, though they are not necessarily referring to the same concepts. In particular, the rules **JavascriptEnabledRule** and **ExternalBehaviorRule** in the system derive from the work of Xu et al. [12], while the other rules (**BlacklistRule**, **HumanTimerRule**, **PagesLoadedRule**, **AcceptLangRule**, **TimePeriodRule**, **UserAgentRule**, **DoNotTrackRule**} e **RedirectTimeRule**}) and their classifications are this work's novel contribution.

Fig. 3 shows the system architecture. The system relies on two parallel processes, each focused on one of the main modules, **Online Analysis** and **Offline Analysis**. This division of the analysis process is also based on the proposal by Kits et Al [7]. The former process is directly responsible for what happens when a HTTP request is received in the server and the procedures that take place before a response is sent back to the requester.

First, the URL requested is checked in the **URL Hash** module, and if it's a valid URL, the request proceeds to the **Online Analysis**. This part of the system passes the received request through various *online rules*. After using the results from the rules' tests to arrive at a conclusion on the legitimacy of the click, the online analysis is completed and the request data, along with the module status and results from rule tests, are stored in the **Database**.

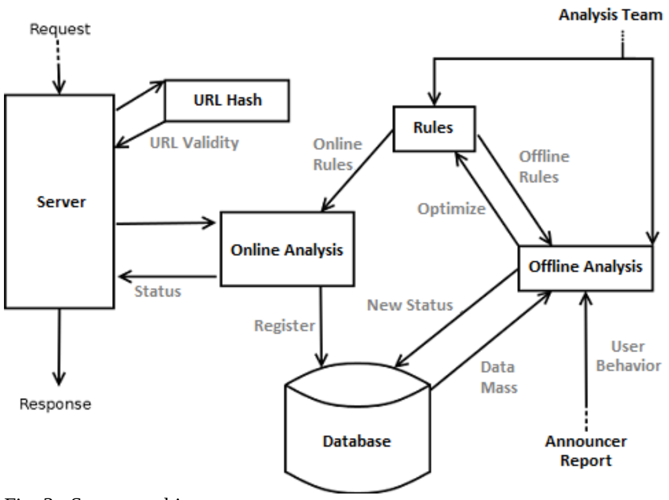


Fig. 3. System architecture.

The **Offline Analysis** process, on the other hand, takes recorded requests from the database as input. The module takes the *offline rules* from the **Rules** module, and, similarly to the other process, runs the input through a series of tests from the rules. After analyzing the requests with this other set of rules, the offline analysis module may change the overall status of the request, working as the final automatic step in determining whether or not a click is malicious. Based on the results, the module may also make changes in rules by adjusting their impact on the overall analysis process, namely with their *weight* parameter. The module can have **Announcer Reports** to help arrive at a conclusion over click status. Those reports focus on recording user behavior on the announcer's web page and are used by the *ExternalBehaviorRule*.

Although the system works automatically without user intervention, it is suggested that a team of security analysts that might work on the system when necessary and would be able to change the rules as they see fit, excluding or including new ones, or reviewing and interfering on the offline analysis module results if they judge it necessary, *e.g.* changing the status of a request in the database.

The way users are redirected towards the desired site was changed to properly collect access information. The process of clicking on an ad and going to the advertiser's site usually works as follows: the user visits the publisher's site, which loads both the page from the publisher's domain and a javascript file from the network. This javascript contains both a link to the network's domain and an image of the ad. The user may click on this image, which redirects them to the network's page, which in turn redirects them to the announcer's site. Our process is very similar, but there are 2 pages in the ad network domain that the user is redirected through, called **adRequest.html** and **redirect.html**. This allows the server to obtain much more information about the user (further details on Section V).

A. Types of Rules

The main method of detecting frauds the system uses are the **Rules**, as previously discussed. The idea is to test every received click with every rule and the system will then be able to determine whether the click is suspicious. Rules are divided in two categories related to the system where they're applied: **online** or **offline**.

- As their tests must be executed in real time and at every HTTP request received, **online rules** are quick and generally less impacting by design. Although it may not make a difference for smaller sites, those with high traffic might find their ability to answer to clients quickly impacted by the system's presence, which should be avoided. Another constraint that comes with these rules is that they must only use information that has been acquired from the user's immediate access, such as the HTTP request received.
- **Offline rules**, in contrast, are only applied over

requests that have already been analyzed by the online rules. This type of rule uses mainly the database, and thus more extensive tests on clicks, like identifying if an IP address loaded all expected files from the domain or looking for unusual access patterns to the ad. This type of rule may also take as long as needed to execute, since its function is not related to the user's experience.

Rules are also classified in two other groups in relation to how critical for the click analysis they are. They are divided between decisive rules and indicative rules.

- If a click fails to pass at a **decisive rule**, it is automatically considered a fraud. These types of rules must be only those that verify aspects that a normal user will never miss out on, such as having HTTP request fields with expected values.
- **Indicative rules** are, then, the rules that give a probability that a click may be illegitimate, which is to say, if it fails at one of these rules, it will not be certain that it is a malicious click. Such rules also have a **weight** factor to each of them. It defines how relevant the rule is for the overall legitimacy of the user and may be adjusted as necessary. There are also negative weights, which mean that failing the rule's test does not affect the user's rating negatively, but passing them improves it.

Rules are described below, starting with the *decisive* and *online* rules, and elaborate on what they are meant to verify along with the design process behind them:

1) BlacklistRule

The system maintains a table with blacklisted IPs that have had too many frauds attributed to them. This rule simply compares the requester's IP address with what's already on the list, and accuses them of fraud if there's a match. It will also block off other unexpected IP addresses, like invalid IP codes or the IP address from the publisher themselves.

2) HumanTimerRule

For this rule, the idea is to see if the user's interaction with the ad is humanly possible, so the time difference between the ad's visualization and the user's click is verified. If they're quicker than a certain threshold, they fail the test. The threshold is set to 0.5 seconds which takes into account average human reaction times, as described below.

Thorpe et al. [14] point to a value of 0.15 seconds for the minimum time between the eyes' visualization and the brain identifying the image, while Census at School Canada [15] and Human Benchmark [16] show similar results. Fig. 4 shows a graph where the highest number of visitors reacted at around 260ms. Since this rule is set to only identify users with inhumanly fast click times, the value decided upon needed to be below the average, as not to accuse real people, and was set for 0.2 seconds.

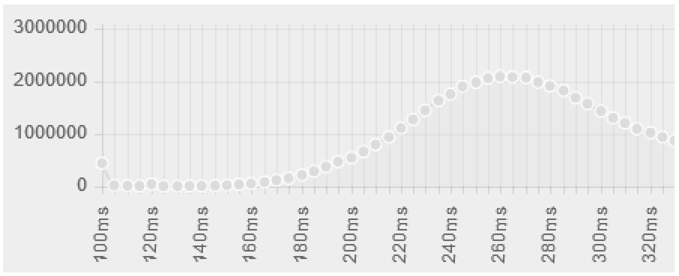


Fig. 4. Quantity of users versus reaction time. Source: [16]

This value alone does not consider the time it would take to realize interest with the content of the ad itself. 0.3 seconds was estimated as the time it would take for the user to realize said interest in the ad. Note that 0.3 seconds for the user's interest is a small amount of time, so it is estimated that the threshold could be changed to higher values, like 1 or even 2 seconds, without blocking off real clicks.

3) AcceptLangRule

As real users are expected to be using web browsers to access the ad, this rule is used to verify the value of the *AcceptLang* header in the HTTP Request. Browsers usually have this field set in the same way for every request sent from it, so the rule will test if the header is valid and not empty. This rule is decisive, in contrast to the *DNTRule* in Subsection IV.A.7, because an empty or otherwise incorrect header field implies that the user isn't using a normal browser at all and is thus most likely not legitimate.

4) PagesLoadedRule

This rule is decisive and *offline*, as it needs to have the log of accesses to the relevant domain pages to be properly executed. The idea is to verify every user's access history in relation to the ad network and see if it's similar to how a normal user is expected to behave. For this, the rule will search for pages that are expected to be loaded, like the ad's image and images present within the ad network's redirecting pages, and also pages that aren't to be loaded, like hidden image links (see Subsection V.A for implementation details). Not having loaded those pages will identify the user as a fake.

Indicative and online rules are:

1) JavascriptEnabledRule

Xu et al. [2] mentions that at least 98% of internet users have javascript enabled and indicates its presence as a sign of click legitimacy. To detect whether or not a given user has it disabled, HTTP cookies are used. To do this, a cookie is set on the first redirecting page of the ad network's domain through a javascript code. Cookie presence is verified on the access to the second page, and if it is indeed sent by the user, they will pass this test.

According to [17] 3.7% of users block cookies and an insignificant amount blocks javascript, while more recently [18] gives a 99.93% usage percentage for javascript and 98% for cookies. This indicates that it's unlikely for a real user to fail this test, but still possible.

2) UserAgentRule

A rule that, in a similar way from the *AcceptLangRule*, verifies if the received requests contain a certain HTTP header; in this case, a valid *UserAgent* field, basically seeing if the value matches what's to be expected from a normal browser. A small weight for this rule is suggested, as it is easy for an attack to merely change the field's value and there's a possibility a real user would be using a browser whose *UserAgent* isn't accounted for in the system. The latter case is also why this rule is defined as indicative, rather than decisive; a user using a browser that's not well-known could have their click blocked in this case.

3) DoNotTrackRule

DoNotTrack or DNT is an HTTP field that's used for browsers in private mode and indicates for the sites being visited not to maintain user information. Since there is no particular reason for an attacker to worry about setting up this field, as they most likely would be using proxies in the first place, the presence of this field is a good way of identifying a legitimate user. Since passing in this test increases the likelihood of legitimacy for the click, and failing it does not indicate suspect behavior, the rule should have a small and *negative weight* (as explained earlier in this section).

4) RedirectTimeRule

While conducting the tests, it was noticed that the time it took for a browser to process a redirect response was much quicker than for the attack bot. The browser could answer the redirect triggered by the "*meta refresh*" field in the HTML page with a value of 0 in 0.5 seconds, the bot used for testing was unable to react in less than 1.1 seconds. This led to the creation of this rule, which looks at the time between accessing the first and second pages from the network domain, and fails whenever the user takes more than the expected time. Although the bot designed and used in tests did not manage to bypass this rule, it should not be decisive, as the redirect time for *meta refresh* will depend on browser, and users with older machines or less used browsers may fail it; even then, a relatively high weight for this rule is suggested.

Finally, the indicative and offline rules are the following:

1) TimePeriodRule

As discussed, one of the ways of identifying abnormal accesses is to check for user behavior that differs significantly from the normal. To achieve that, this rule looks over the received requests on the database and finds odd patterns. Constant and quick accesses are clear fraudulent behavior, and in order to defend against that, two ways of identifying such patterns that this rule checks for are defined:

- The first is to look for three or more clicks by the same IP in under a short period of time (we opted for 30 seconds, but the value could be changed).
- The other method is to search for five or more clicks within a longer period of time, (10 minutes, but again changeable) and check if the interval between them is fairly constant.

Failing either or both of those tests leads to failing the rule, which is decisive because it verifies for extreme behavior that is very unlikely to happen with real clicks.

2) ExternalBehaviorRule

Behavior logs are an important part of fraud detection systems. One issue that arises from designing a defense system for an ad network is that it doesn't have access to the user's actions once they are within the advertiser's domain, and thus we're unable to collect information based on aspects that are significantly hard to forge, like mouse movements and time spent on pages.

For this rule we count on the support of the advertiser, who is motivated to help the ad network's defense, as advertisers, who in theory pay for publisher services and ad networks will often suffer the biggest losses from fraudulent attacks. Thus, the advertiser will log certain aspects of the visit from any user that comes from the ad network's domain, and organize this data to send it back to the ad network in the format shown by Table I.

TABLE I
USER BEHAVIOR REPORT FROM THE ADVERTISER

Category	Fields
Clicks	First page
	Other pages
Mouse Scrolls	First page
	Other pages
Mouse Events	First page
	Other pages
Time spent	First page
	Other pages
Visited pages	Total

All categories in the table except for the last one should be stored in 2 separate counts each: one for the *initial page*, and another for the *other pages* in the site. A user that's truly interested will likely have high values in most of the fields collected, while a fraudulent one will typically not interact much with any pages, or only with the first one.

V. Implementation

THE whole project was coded in Python 3.6, and the persistent data was stored in a database in PostgreSQL 9.6.

As already mentioned, the problem was approached by looking at it from both the attack and defense sides: first a basic form of click fraud was implemented, and how to defend against it. Then, an attack to surpass this new defense, and a new method of detection for this newest attack type, and so on.

A. Defense

The main method of detecting frauds the system uses are the **Rules**, as previously discussed. The idea is to test every received click with every rule and the system will then be able to determine whether the click is suspicious. By its turn, rules fall into two categories related to the system where they're applied: **online** or **offline**.

The first part of the system that requests pass through is the URL Hash, whose purpose is to prevent people from using the same ad multiple times without accessing a publisher's site. When the ad's javascript is loaded, the URL presented to the user will depend on identifiers the system gets from their HTTP Request, like IP and UserAgent field. Those values will be hashed together with a private key, which should be changed periodically, to create a unique URL for the user. When the request for the ad's URL is received, the hash is recomputed and compared with the one in the URL. If they match, the user will access the normal page as usual. Otherwise, they will receive an error page. This whole process is illustrated in Figure 5.

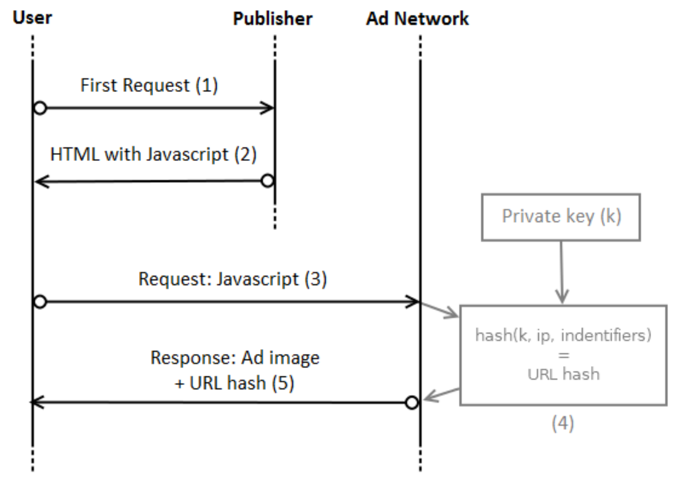


Fig. 5. Process to give the user a unique ad URL.

The user will then go through pages as already mentioned, **adRequest.html** and **redirect.html**. This is done to obtain more information than would be possible from a single access, although each of the pages verifies different things on the user's access:

- In the first page access, the request is passed through the **HumanTimer**, **Blacklist**, **AcceptLang** and **DNT** rules. **HumanTimerRule** is applied here since it assesses the time between the request for the ad's data and the click on it, while the other rules are applied here since they already have all the information necessary from a single request.
- The rules **UserAgent**, **Javascript** and **RedirectTime** are used on the **redirect.html** page's request. These tests need or work better when executed with the information obtained from both user's requests.

The navigation that goes for the user within the ad network's domain, integral part of obtaining the information used by the rules, works as follows: After clicking on the ad with a valid URL, the user is redirected to the network's domain, namely to *adRequest.html*. This page, besides redirecting the user to the next page automatically, also sets a cookie and has a 1-pixel image to be loaded in it. On the second page, the cookie's presence is verified, and another 1-pixel image is present, except this one is hidden, in a way that a normal browser would not load it. This second page redirects the user to the advertiser's site. The cookie's importance for the rule *JavascriptEnabledRule* was discussed in Section IV, while the images are there for the *PagesLoadedRule*.

To connect both page requests as being from the same user, every access to the first page is stored, and when an access to the second page is received, it is matched with the earliest request with the same identifiers, such as the IP, UserAgent and AcceptLang fields. It is also checked if the time between these accesses is short enough; in this case, 3 seconds was established to be the maximum period of time between loading both pages for a normal user.

Even if the defense system decides that a user is committing fraud, it will still work normally, sending the requester to the desired page. This happens for two main reasons:

- Resilience against trial-and-error methods. If the system blocked every suspicious user and access, attackers would be able to use this info to verify if their attempts were successful or not, and make improvements to their attack methods accordingly.
- Reduce the impact of false-positives. The system may wrongly accuse a normal user of being malicious; in this case, we don't want the user to be unable to go through with their browsing, as that would also hurt the advertiser who'd lose a possible client.

Relatedly, when reporting to the announcer the system's results, the ad network should avoid specifying which clicks exactly were identified as fraud as suggested by Kitts et al. [7], and instead provide the percentage or the quantity of blocked clicks. This turns the system even more resistant against malicious users, which may include advertisers interested in attacking the ad network's system for personal gain, while still giving legit advertisers a report on how well the service is working.

B. Attack

The bot created to test the system focuses on getting the highest number of clicks possible without being caught by the defense. It was developed in Python 3.6 like the defense system, and mainly uses the "*requests*" Python library to access the agents' sites as a client. At first the bot is relatively simple in its access strategy, which can be seen in Figure 6. It will initially access the publisher site normally, and once it obtains the ad's URL, it will call for bots in parallel to quickly access the ad many times, and those bots follow the normal navigation path independently from each.

To emulate the progress a real attacker would go through while learning how the defense system works, the bot has different configurations of **modules**, which alter how the bot goes through with its attacks. Some modules are counterparts to the rules presented, and they are:

- **NormalAccessModule**: bot will access the publisher's site normally to obtain only valid ad addresses. This was the first module to be implemented, as without it the bot won't even be able to access the pages that will redirect it to the advertiser's site.
- **HumanTimerModule**: the program now waits a reasonable and humanly possible time before executing each click (roughly 0.6 seconds). This prevents detection from techniques that would notice the time between each click is unrealistically short for a normal person.
- **HeadersModule**: uses legitimate, pre-defined HTTP headers to simulate what would be expected from a user with a browser.

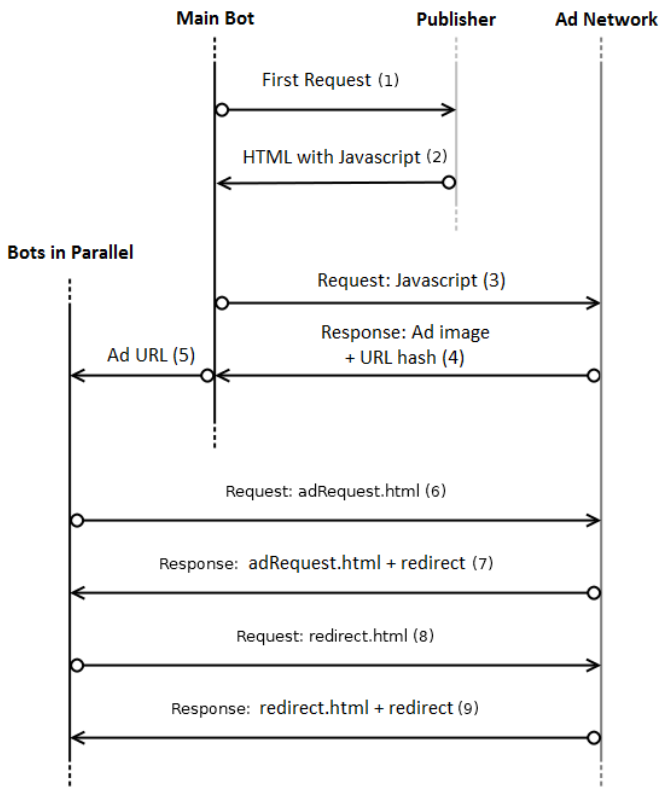

Main Bot
Publisher
Ad Network
First Request (1)
HTML with Javascript (2)
Bots in Parallel
Request: Javascript (3)
Response: Ad image + URL hash (4)
Ad URL (5)
Request: adRequest.html (6)
Response: adRequest.html + redirect (7)
Request: redirect.html (8)
Response: redirect.html + redirect (9)

Fig. 6. Basic bot setup.

- **DNTModule**: adds a valid DoNotTrack HTTP header to the bot's requests. This module is separate from the previous one because, while the headers from the previous module are expected from every user, a header such as DNT is optional and only used by normal users in certain circumstances.
- **WideAccessModule**: the bot will access every link passed by the HTML pages received from the ad network domain. This is meant to be a measure against a defense technique that checks if the user has accessed expected links and files, given that many bots will not load things such as images, since they're useless for their objectives.
- **SelectiveAccessModule**: an extension to the *WideAccessModule*, which prevents the bot from accessing links a normal user would have no access to, such as images hidden in the HTML code. This further enhances the bot's attempt to browse as a normal user with a browser would.
- **RandomTimeModule**: adds a pseudo-random amount of time between clicks to turn pattern recognition more difficult;
- **CookieModule**: this module makes the bot store cookies sent by the pages correctly;
- **CompleteAccessModule**: further extending on the *SelectiveAccessModule*, this module makes the bot load all pages in ways that are expected from a normal user, such as loading ad images and page icons. Those types of files may be ignored by less advanced configurations, since they aren't part of any of the ad network's domain's pages, or part of the click process.

Notice that many of the modules were done with knowledge that an attacker might take a long period of time to discover if specifically targeting the system, such as figuring out if the system is blocking off their clicks because the time between each access is too short.

VI. Tests and Results

To test the defense system, the developed bot was put against it, with different combinations of modules enabled, and finally with all modules active at once. Ultimately the system could identify all the attack attempts as frauds, but a low-frequency attack attempt would have been able to go undetected, as the system's rules don't individually address infrequent but continuous fraud attempts (more on that in the final section, VII).

Tests were run on a Windows 10 operational system. The servers that represent the three online advertisement agents (publisher, ad network and advertiser) and the bot all ran in a local network, using different ports each to simulate the different site domains that would be required for the testing. The three agent's servers work in a normal way, being able to respond to any HTTP Request of the GET type that they receive, displaying their respective pages.

The bot's attacks are called 6 times, each time with a different set of modules, which will from now on be referred to as "configurations". These configurations are defined by simple text files with values 1 ("on") or 0 ("off") for all possible modules; this is done to represent the degree of complexity of the simulated attacks. Table II shows how the configurations are all set, with the value "1" meaning that the module is activated, while "0" meaning the opposite, and the roman numerals representing each configuration.

TABLE II
MODULES PRESENT IN EACH CONFIGURATION

Module	I	II	III	IV	V	VI
NormalAccessModule	1	1	1	1	1	1
HumanTimerModule	1	0	1	1	1	1
HeadersModule	0	1	1	1	1	1
DNTModule	0	1	1	1	1	1
WideAccessModule	0	0	1	1	1	1
SelectiveAccessModule	0	0	0	1	1	1
CookieModule	0	0	0	0	1	1
CompleteAccessModule	0	0	0	0	0	1
RandomTimeModule	0	0	0	0	0	1

For the bot itself, three parallel threads are used for every attack attempt. Given that each access is registered as a new click, this would lead to better and quicker results for the attacker, as explained before in Subsection V.B.

Table III shows the chosen weight values for all the indicative rules. Those values aren't definitive or optimal, but were picked in a way that establishes the relative importance of each indicative rule if compared with the others.

TABLE III
WEIGHTS USED FOR EACH RULE DURING TESTING

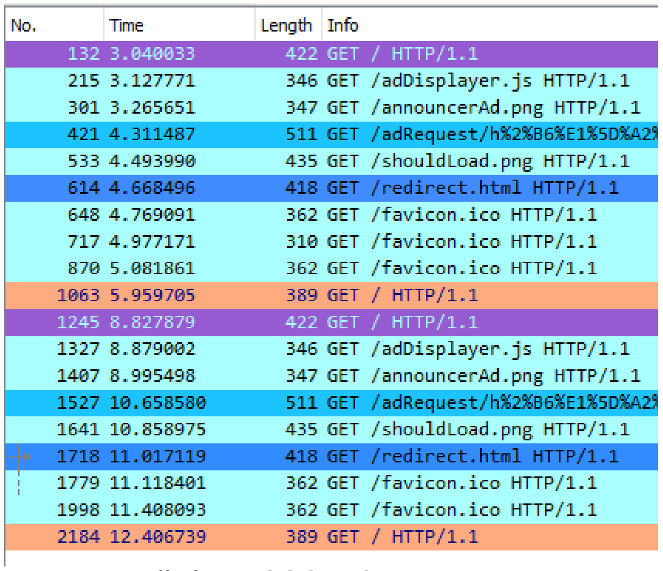
Rule	Weight
JavascriptEnabledRule	2.0
UserAgentRule	2.0
RedirectTimeRule	3.0
DNTRule	-1.0
BehaviorRule	3.0
TimePeriodRule	2.0

The first part of the test was to see if the system would incorrectly label a real user as a fraud, and for that the click process was done twice through the Firefox 59.0.1 (64-bit) browser. Figure 7 shows the traffic captured between the servers during the 2 accesses. For this and the next images showing the traffic captures, the following color code is used:

- Purple captures are from the publisher's site.
- Blue captures show the ad network.
- The darker blue tones show the 2 accesses that configure a click, namely for the pages adRequest.html and redirect.html.
- Finally, the advertiser's domain is represented by orange.

Table IV shows the details of some of the packets sent to the ad network's server during the click process, and Table V shows results for every rule test the click went through. A value of 't' means it passed the test, while 'f' means failure. The click will be considered a fraud if it either fails a decisive rule or gets a score below 0.5. As they show, the real user passed through all online tests without issues.

On the first attack, seen in Figure 8, note that bots are identified as frauds as soon as they access the first page on the network's domain. Given that in this configuration the bot does not use expected HTTP headers, the bots fail on the *AcceptLangRule*, as Tables VI and VII show. With this attack's report and the following ones, only one of the clicks' status is shown in the table unless stated otherwise, as the 3 parallel clicks will, for the most part, have the same exact results.



```
No.        Time          Length Info
     132 3.040033          422 GET / HTTP/1.1
     215 3.127771          346 GET /adDisplayer.js HTTP/1.1
     301 3.265651          347 GET /announcerAd.png HTTP/1.1
     421 4.311487          511 GET /adRequest/h%2%B6%E1%5D%A2%
     533 4.493990          435 GET /shouldLoad.png HTTP/1.1
     614 4.668496          418 GET /redirect.html HTTP/1.1
     648 4.769091          362 GET /favicon.ico HTTP/1.1
     717 4.977171          310 GET /favicon.ico HTTP/1.1
     870 5.081861          362 GET /favicon.ico HTTP/1.1
    1063 5.959705          389 GET / HTTP/1.1
    1245 8.827879          422 GET / HTTP/1.1
    1327 8.879002          346 GET /adDisplayer.js HTTP/1.1
    1407 8.995498          347 GET /announcerAd.png HTTP/1.1
    1527 10.658580         511 GET /adRequest/h%2%B6%E1%5D%A2%
    1641 10.858975         435 GET /shouldLoad.png HTTP/1.1
    1718 11.017119         418 GET /redirect.html HTTP/1.1
    1779 11.118401         362 GET /favicon.ico HTTP/1.1
    1998 11.408093         362 GET /favicon.ico HTTP/1.1
    2184 12.406739         389 GET / HTTP/1.1
```

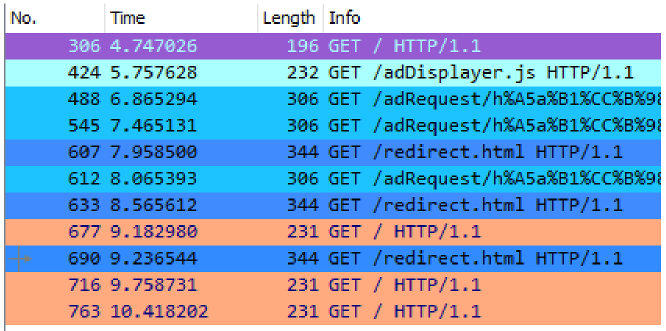
Fig. 7. HTTP Traffic from 2 ad clicks via browser.

TABLE IV
REQUESTS FROM THE BROWSER

id	*time*	*path*	*cookies*	*status*	*report_id*
9	10:48:22.06	/adDisplayer.js		valid	
10	10:48:22.18	/announcerAd.png		valid	
12	10:48:23.84	/adRequest.html		valid	2
11	10:48:24.04	/shouldLoad.png		valid	
14	10:48:24.20	/redirect.html	JSEnabled=true	valid	2
13	10:48:24.30	/favicon.ico		valid	
15	10:48:24.59	/favicon.ico		valid	

TABLE V
SYSTEM RESULTS FOR A REAL USER

id	**dnt**	**java**	**human**	**list**	**agent**
2	t	t	t	t	t
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
t	t				1.17



```
No.        Time          Length Info
     306 4.747026          196 GET / HTTP/1.1
     424 5.757628          232 GET /adDisplayer.js HTTP/1.1
     488 6.865294          306 GET /adRequest/h%A5a%B1%CC%B%98
     545 7.465131          306 GET /adRequest/h%A5a%B1%CC%B%98
     607 7.958500          344 GET /redirect.html HTTP/1.1
     612 8.065393          306 GET /adRequest/h%A5a%B1%CC%B%98
     633 8.565612          344 GET /redirect.html HTTP/1.1
     677 9.182980          231 GET / HTTP/1.1
     690 9.236544          344 GET /redirect.html HTTP/1.1
     716 9.758731          231 GET / HTTP/1.1
     763 10.418202         231 GET / HTTP/1.1
```

Fig. 8. HTTP Traffic from the first bot configuration.

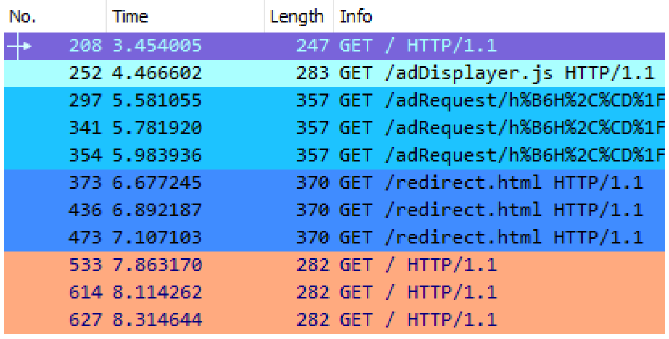
TABLE VI
REQUESTS FROM THE FIRST CONFIGURATION

id	*time*	*path*	*cookies*	*status*	*report*
16	11:40:01.66	/adDisplayer.js		valid	
17	11:40:02.77	/adRequest.html		fraud	3
19	11:40:03.37	/adRequest.html		fraud	4
18	11:40:03.86	/redirect.html	JSEnabled=true;	fraud	3
21	11:40:03.97	/adRequest.html		fraud	5
20	11:40:04.47	/redirect.html	JSEnabled=true;	fraud	4
22	11:40:05.14	/redirect.html	JSEnabled=true;	fraud	5

TABLE VII
REQUESTS FROM THE FIRST CONFIGURATION

id	**dnt**	**java**	**human**	**list**	**agent**
3	f		t	t	
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
	f				0.00

With the bot's second configuration, attack attempts in Figure 9 were once again identified by the system. In this case, the first bot managed to pass through the first page's tests, but failed for rules *JavascriptRule* and *RedirectTime* right after. The other 2 bots were identified in the first page thanks to the *HumanTimerRule*, as they accessed the ad link way too quickly for it to be done by a legitimate, human user. The reports are present at Tables VIII and IX. Note that in this case the report on all 3 accesses is shown.



```
No.    Time       Length Info
208 3.454005      247 GET / HTTP/1.1
252 4.466602      283 GET /adDisplayer.js HTTP/1.1
297 5.581055      357 GET /adRequest/h%B6H%2C%CD%1F
341 5.781920      357 GET /adRequest/h%B6H%2C%CD%1F
354 5.983936      357 GET /adRequest/h%B6H%2C%CD%1F
373 6.677245      370 GET /redirect.html HTTP/1.1
436 6.892187      370 GET /redirect.html HTTP/1.1
473 7.107103      370 GET /redirect.html HTTP/1.1
533 7.863170      282 GET / HTTP/1.1
614 8.114262      282 GET / HTTP/1.1
627 8.314644      282 GET / HTTP/1.1
```

Fig. 9. HTTP Traffic from the second bot configuration.

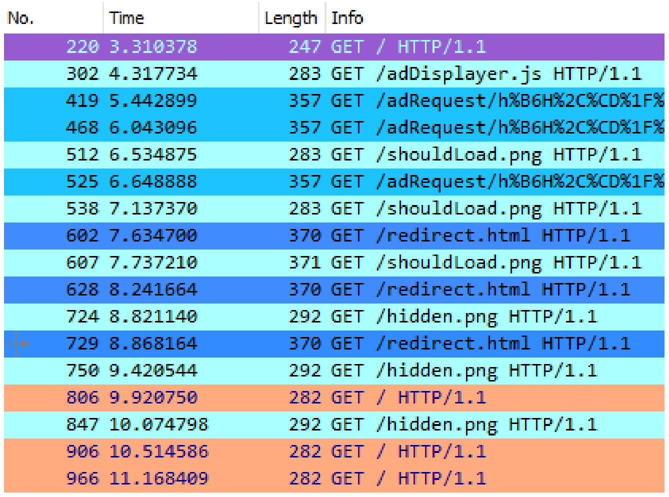
TABLE VIII
REQUESTS FROM THE SECOND CONFIGURATION

id	*time*	*path*	*cookies*	*status*	*report_id*
23	13:04:31.04	/adDisplayer.js		valid	
24	13:04:32.16	/adRequest.html		fraud	6
26	13:04:32.36	/adRequest.html		fraud	7
28	13:04:32.56	/adRequest.html		fraud	8
25	13:04:33.25	/redirect.html		fraud	6
27	13:04:33.47	/redirect.html		fraud	7
29	13:04:33.68	/redirect.html		fraud	8

TABLE IX
REQUESTS FROM THE SECOND CONFIGURATION

id	**dnt**	**java**	**human**	**list**	**agent**
6	t	f	t	t	t
7	t		f	t	
8	t		f	t	
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
f	t				0.43
	t				0.00
	t				0.00

In the following test, the bot used the following modules: *NormalAccess*, *HumanTimer*, *Headers*, *DNT* and *WideAccess*. Figure 10 shows the traffic, and from that it can be noted the difference that the *WideAccessModule* introduces: the bot now sends requests for the images in the network's pages, including the ones that a normal user wouldn't (*hidden.png*). However, the bot does not pass the tests, as in Tables X and XI. It fails to pass rules *JavascriptEnabled* and *RedirectTime*, which puts the score just under 0.5.



```
No.    Time        Length Info
220 3.310378       247 GET / HTTP/1.1
302 4.317734       283 GET /adDisplayer.js HTTP/1.1
419 5.442899       357 GET /adRequest/h%B6H%2C%CD%1F%
468 6.043096       357 GET /adRequest/h%B6H%2C%CD%1F%
512 6.534875       283 GET /shouldLoad.png HTTP/1.1
525 6.648888       357 GET /adRequest/h%B6H%2C%CD%1F%
538 7.137370       283 GET /shouldLoad.png HTTP/1.1
602 7.634700       370 GET /redirect.html HTTP/1.1
607 7.737210       371 GET /shouldLoad.png HTTP/1.1
628 8.241664       370 GET /redirect.html HTTP/1.1
724 8.821140       292 GET /hidden.png HTTP/1.1
729 8.868164       370 GET /redirect.html HTTP/1.1
750 9.420544       292 GET /hidden.png HTTP/1.1
806 9.920750       282 GET / HTTP/1.1
847 10.074798      292 GET /hidden.png HTTP/1.1
906 10.514586      282 GET / HTTP/1.1
966 11.168409      282 GET / HTTP/1.1
```

Fig. 10. HTTP Traffic from the third bot configuration.

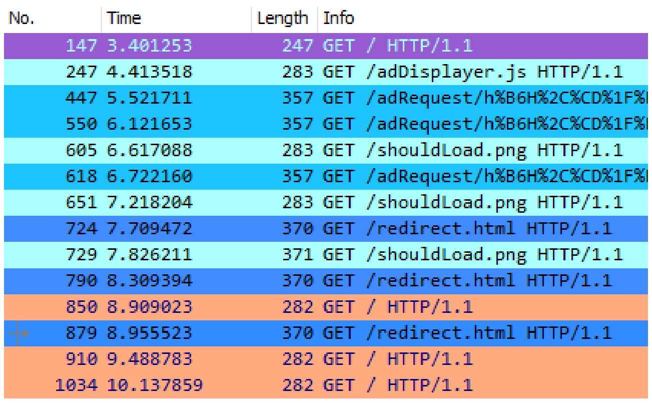
TABLE X
REQUESTS FROM THE THIRD CONFIGURATION

id	*time*	*path*	*cookies*	*status*	*report_id*
30	13:21:55.77	/adDisplayer.js		valid	
33	13:21:56.90	/adRequest.html		fraud	9
36	13:21:57.50	/adRequest.html		fraud	10
31	13:21:57.99	/shouldLoad.png		valid	
39	13:21:58.10	/adRequest.html		fraud	11
32	13:21:58.59	/shouldLoad.png		valid	
34	13:21:59.09	/redirect.html		fraud	9
35	13:21:59.19	/shouldLoad.png		valid	
37	13:21:59.70	/redirect.html		fraud	10
38	13:22:00.28	/hidden.png		valid	
40	13:22:00.32	/redirect.html		fraud	11
41	13:22:00.87	/hidden.png		valid	
42	13:22:01.53	/hidden.png		valid	

TABLE XI
REQUESTS FROM THE THIRD CONFIGURATION

id	dnt	java	human	list	agent
9	t	f	t	t	t
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
f	t				0.43

Verifying the results from Table XIII it is noticed that the scores and overall report from the system are very similar, contrasting with the traffic differences seen in Figure 11 and Table XII, which derive from the *SelectiveAccessModule*. This happens because, even with the new module to improve navigation, the attack still doesn't take any measures against the *JavascriptEnabled* and *RedirectTime* rules.



```
No.    Time       Length Info
 147 3.401253     247 GET / HTTP/1.1
 247 4.413518     283 GET /adDisplayer.js HTTP/1.1
 447 5.521711     357 GET /adRequest/h%B6H%2C%CD%1F%
 550 6.121653     357 GET /adRequest/h%B6H%2C%CD%1F%
 605 6.617088     283 GET /shouldLoad.png HTTP/1.1
 618 6.722160     357 GET /adRequest/h%B6H%2C%CD%1F%
 651 7.218204     283 GET /shouldLoad.png HTTP/1.1
 724 7.709472     370 GET /redirect.html HTTP/1.1
 729 7.826211     371 GET /shouldLoad.png HTTP/1.1
 790 8.309394     370 GET /redirect.html HTTP/1.1
 850 8.909023     282 GET / HTTP/1.1
 879 8.955523     370 GET /redirect.html HTTP/1.1
 910 9.488783     282 GET / HTTP/1.1
1034 10.137859    282 GET / HTTP/1.1
```

Fig. 11. HTTP Traffic from the fourth bot configuration.

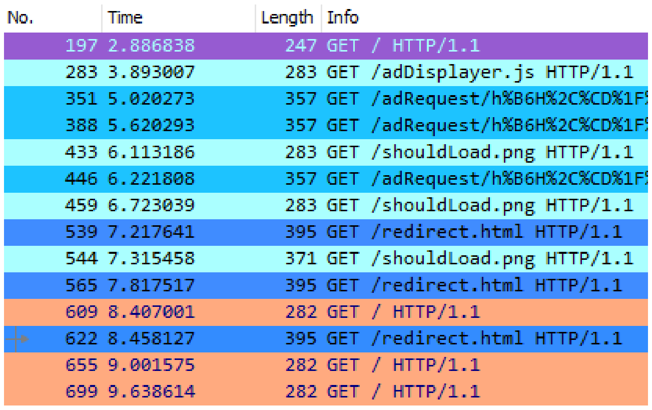
TABLE XII
REQUESTS FROM THE FOURTH CONFIGURATION

id	*time*	*path*	*cookies*	*status*	*report_id*
43	16:55:27.59	/adDisplayer.js		valid	
46	16:55:28.70	/adRequest.html		fraud	12
49	16:55:29.30	/adRequest.html		fraud	13
44	16:55:29.79	/shouldLoad.png		valid	
51	16:55:29.90	/adRequest.html		fraud	14
45	16:55:30.39	/shouldLoad.png		valid	
47	16:55:30.89	/redirect.html		fraud	12
48	16:55:31.00	/shouldLoad.png		valid	
50	16:55:31.49	/redirect.html		fraud	13
52	16:55:32.13	/redirect.html		fraud	14

TABLE XIII
REQUESTS FROM THE FOURTH CONFIGURATION

id	dnt	java	human	list	agent
12	t	f	t	t	t
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
f	t				0.43

From the fifth configuration onwards, the online analysis alone could not identify the fraud, which can be visualized in the report from Table XV. In the report, row '15' shows the results before the offline analysis acted, and the '15*' row is for the results after. In this configuration, the attacker uses a *CookieModule* and reads through the javascript code to bypass the *JavascriptEnabledRule*. However, it still fails the *RedirectTime* rule, and is reported as fraud by both the *PagesLoaded* and *TimePeriod* offline rules.



```
No.    Time       Length Info
 197 2.886838     247 GET / HTTP/1.1
 283 3.893007     283 GET /adDisplayer.js HTTP/1.1
 351 5.020273     357 GET /adRequest/h%B6H%2C%CD%1F
 388 5.620293     357 GET /adRequest/h%B6H%2C%CD%1F
 433 6.113186     283 GET /shouldLoad.png HTTP/1.1
 446 6.221808     357 GET /adRequest/h%B6H%2C%CD%1F
 459 6.723039     283 GET /shouldLoad.png HTTP/1.1
 539 7.217641     395 GET /redirect.html HTTP/1.1
 544 7.315458     371 GET /shouldLoad.png HTTP/1.1
 565 7.817517     395 GET /redirect.html HTTP/1.1
 609 8.407001     282 GET / HTTP/1.1
 622 8.458127     395 GET /redirect.html HTTP/1.1
 655 9.001575     282 GET / HTTP/1.1
 699 9.638614     282 GET / HTTP/1.1
```

Fig. 12. HTTP Traffic from the fifth bot configuration.

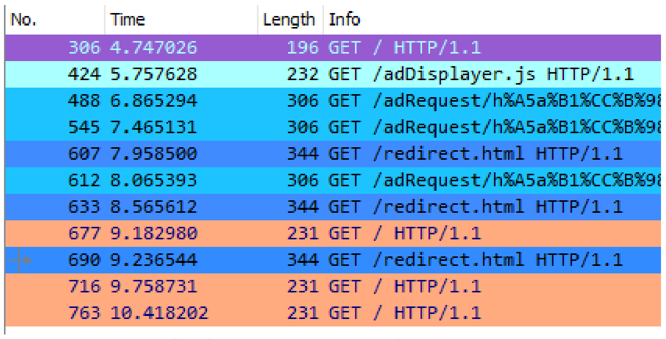
TABLE XIV
REQUESTS FROM THE FIFTH CONFIGURATION

id	*time*	*path*	*cookies*	*status*	*report_id*
53	17:06:56.68	/adDisplayer.js		valid	
56	17:06:57.80	/adRequest.html		fraud	15
59	17:06:58.40	/adRequest.html		fraud	16
54	17:06:58.90	/shouldLoad.png		valid	
61	17:06:59.00	/adRequest.html		fraud	17
55	17:06:59.51	/shouldLoad.png		valid	
57	17:07:00.00	/redirect.html	JSEnabled=true;	fraud	15
58	17:07:00.10	/shouldLoad.png		valid	
60	17:07:00.60	/redirect.html	JSEnabled=true;	fraud	16
62	17:07:01.24	/redirect.html	JSEnabled=true;	fraud	17

TABLE XV
REQUESTS FROM THE FIFTH CONFIGURATION;
BEFORE AND AFTER OFFLINE ANALYSIS

id	dnt	java	human	list	agent
15	t	t	t	t	t
15*	t	t	t	t	t
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
f	t				0.71
f	t		f	f	0.42

For the final test, the bot has all modules activated, and its traffic capture is in Figure 13. As is reported inTables XVI and XVII, the bot passes through the *PagesLoadedRule* now, but is still detected by both *TimePeriod* and *RedirectTime* rules, giving the clicks a fraud status.



```
No.    Time       Length Info
 306 4.747026     196 GET / HTTP/1.1
 424 5.757628     232 GET /adDisplayer.js HTTP/1.1
 488 6.865294     306 GET /adRequest/h%A5a%B1%CC%B%9
 545 7.465131     306 GET /adRequest/h%A5a%B1%CC%B%9
 607 7.958500     344 GET /redirect.html HTTP/1.1
 612 8.065393     306 GET /adRequest/h%A5a%B1%CC%B%9
 633 8.565612     344 GET /redirect.html HTTP/1.1
 677 9.182980     231 GET / HTTP/1.1
 690 9.236544     344 GET /redirect.html HTTP/1.1
 716 9.758731     231 GET / HTTP/1.1
 763 10.418202    231 GET / HTTP/1.1
```

Fig. 13. HTTP Traffic from the sixth bot configuration.

TABLE XVI
REQUESTS FROM THE SECOND CONFIGURATION

id	*time*	*path*	*cookies*	*status*	*report_id*
63	17:12:36.13	/adDisplayer.js		valid	
64	17:12:37.24	/announcerAd.png		valid	
66	17:12:38.33	/adRequest.html		fraud	18
65	17:12:39.42	/shouldLoad.png		valid	
67	17:12:40.52	/redirect.html	JSEnabled=true;	fraud	18
68	17:12:41.70	//favicon.ico		valid	
70	17:12:42.01	/adRequest.html		fraud	19
69	17:12:43.11	/shouldLoad.png		valid	
71	17:12:44.20	/redirect.html	JSEnabled=true;	fraud	19
72	17:12:45.38	//favicon.ico		valid	
74	17:12:49.74	/adRequest.html		fraud	20
73	17:12:50.83	/shouldLoad.png		valid	
75	17:12:51.93	/redirect.html	JSEnabled=true;	fraud	20
76	17:12:53.12	//favicon.ico		valid	

TABLE XVII
REQUESTS FROM THE SIXTH CONFIGURATION

id	**dnt**	**java**	**human**	**list**	**agent**
18	t	t	t	t	t
redir	**lang**	**behav**	**pages**	**tperiod**	**score**
f	t		t	f	0.42

Finally, after running all the tests it is clear that the system managed to correctly detect every fraud attempt from the attack simulation. Although the attacks managed to bypass more rules each time, the high weights of the *RedirectTime* and *TimePeriod* rules were key in identifying the frauds correctly. This also goes to show the importance of weights: if, for example, *RedirectTime* had a weight of 2 instead of 3, and *TimePeriod* had a weight of 1 instead of 2, the sixth and last attack configuration would have had a score higher than 0.5, and thus wouldn't have been identified as fraud.

VII. CONCLUSION

RESULTS obtained match the expectations. Although not yet tested in a real environment, the system has shown good performance against different types of attack, with at least two of the rules identifying each of those attempted frauds. All the attacks attempted by the bot user were identified, but it is emphasized the importance of selecting appropriate values for rules' weights, as different values for key rules would have led to the system not correctly classifying attacks as such. The chosen weights for our tests aren't the only possible values and in a real world scenario through experimentation would be necessary.

Low-frequency attacks are still a threat as malicious clicks spread over long time intervals will most likely not be automatically detected, and if done in a large scale with multiple IP addresses could affect the agents protected by the system. There are however issues with those methods of attack themselves, namely obtaining access to this significant number of IPs and obtaining economical return from simpler attempts of the attack, given the low return for a single click.

Even with the successful elaboration of the system proposed, click fraud still poses challenges for defense systems. For instance, lack of public data sets about click-related subjects, in particular samples of previously identified click frauds, and, in the case of defense systems for ad networks in particular, having no reliable access to user behavior parameters such as mouse movements or time spent on different pages within a domain, essential data for detecting abnormalities with site accesses, given how such things carry inherently human characteristics to them and would be very challenging to replicate with a fake user.

Although this work focused on click fraud detection, the approach adopted is not entirely dependent on the attack specifics and might be extended to click bot detection in general with a subset of the rules used.

A. Future Work

This is ongoing work and there are several possibilities for improvement. Some of them are presented below.

1) LoadingBehaviorRule

One type of rule that could still be added is a verification of user behavior during the loading period that's spent in the two domain pages, before the user is redirect to the advertiser, and would consist of verifying for activities such as mouse events and mouse movements. This rule would have a negative weight, as described on Section IV, given that a normal user might fail the test simply by not interacting with the computer during the brief loading period, but passing the rule does indicate that the user is likely legitimate.

To store the relevant information about the users' accesses, a simpler version of the report described for the rule ExternalBehaviorRule might be used, as seen on Table XVIII.

TABLE XVIII
REPORT TO BE USED FOR LOADINGBEHAVIORRULE

Category	**Fields**
Mouse Moves	First page
	Second page
Mouse Events	First page
	Second page

2) Browser Functionality Detection

Another feature that could enhance detection is to look for browser functionalities on users. Bots generally aren't implemented with browsers. To reminisce, one of the most common click fraud methods involves infecting computers of internet users, and sending instructions to the infected machines so they can operate on the background without knowledge of the computer's owner. This is ideally implemented with light-weight and quick programs, so the person using the computer doesn't notice it.

Based on that and the report from Xu et al. [2], the idea is to make use of a list of functionalities considered universal in various different recent browsers, covering 5 common browsers (namely, *Chrome*, *Firefox*, *Internet Explorer*, *Safari*, *Opera*).

3) Browser Fingerprinting

This is similar to the technique shown in the previous subsection, in that it revolves around sending requests to a user and asking for them, or more specifically their browser, to complete certain tasks, such as rendering a three-dimensional object. The answer can provide information that differs between unique users and browsers, and thus is useful for authentication and identifying malicious users.

Relatedly, there is also *system fingerprinting*. With a similar focus, this field tries to identify distinct users based on their operational system and its characteristics. Adding such techniques would be a viable way of identifying even complex frauds that are being carried out by a single machine, or many distinct machines being used repeatedly, for example.

4) Machine Learning

The system was already made with possible machine learning implementations in mind for the future. Many aspects can be optimized and automatized, such as individual rule weights and rule configurations, which could be tested against datasets of already identified traffic, including frauds, and even lead to discovering configurations that work better in specific situations over others. Even more advanced algorithms could experiment with identifying patterns in the datasets themselves and propose new rules to be used.

5) Real-World Scenarios

Finally, modifying the system for **real-world scenarios** should be considered, as the code developed, although made with scalability in mind, was not tested against massive request quantities. This could present performance and maintainability issues.

Along with this, there may be other problems that were not properly addressed throughout the development of the system, and future, more advanced attacks that the system would not be prepared to defend automatically against might be created and utilized.

REFERENCES

[1] C. McNair, "US ad spending: eMarketer's updated estimates and forecast for 2017," Sep. 2017.

[2] H. Xu, D. Liu, A. Koehl, H. Wang, and A. Stavrou, "Click fraud detection on the advertiser side," in *European Symposium on Research in Computer Security*. 1em plus 0.5em minus 0.4em Springer, 2014, pp. 419–438. doi: 10.1007/978-3-319-11212-1_24

[3] N. Daswani, C. Mysen, V. Rao, S. Weis, K. Gharachorloo, and S. Ghosemajumder, "Online advertising fraud," *Crimeware: understanding new attacks and defenses*, vol. 40, no. 2, pp. 1–28, 2008.

[4] R. J. Oentaryo, E.-P. Lim, M. Finegold, D. Lo, F. Zhu, C. Phua, E.-Y. Cheu, G.-E. Yap, K. Sim, M. N. Nguyen *et al.*, "Detecting click fraud in online advertising: a data mining approach." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 99–140, 2014

[5] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," Internet Requests for Comments, RFC Editor, RFC 7231, June 2014, http://www.rfc-editor.org/rfc/rfc7231.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7231.txt

[6] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Authentication," Internet Requests for Comments, RFC Editor, RFC 7235, June 2014, http://www.rfc-editor.org/rfc/rfc7235.txt . [Online]. Available: http://www.rfc-editor.org/rfc/rfc7235.txt

[7] B. Kitts, J. Y. Zhang, G. Wu, W. Brandi, J. Beasley, K. Morrill, J. Ettedgui, S. Siddhartha, H. Yuan, F. Gao *et al.*, "Click fraud detection: adversarial pattern recognition over 5 years at microsoft," in *Real World Data Mining Applications*. 1em plus 0.5em minus 0.4em Springer, 2015, pp. 181–201. doi: 10.1007/978-3-319-07812-0_10

[8] N. Daswani and M. Stoppelman, "The anatomy of clickbot. a," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. 1em plus 0.5em minus 0.4em USENIX Association, 2007, pp. 11–11.

[9] J. Leyden, "Botnet implicated in click fraud scam," May 2006. [Online]. Available: https://www.theregister.co.uk/2006/05/15/google_adword_scam

[10] United States District Court, "Microsoft vs Eric Lam et. al." 2009.

[11] P. Pearce, C. Grier, V. Paxson, V. Dave, D. McCoy, G. M. Voelker, and S. Savage, "The zeroaccess auto-clicking and search-hijacking click fraud modules," California Univ Berkeley Dept of Electrical Engineering and Computer Sciences, Tech. Rep., 2013.

[12] C. P. Avila and M. Vijaya, "Click through rate prediction for display advertisement," *International Journal of Computer Applications*, vol. 136, no. 1, 2016.

[13] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin *et al.*, "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1em plus 0.5em minus 0.4em ACM, 2013, pp. 1222–1230. doi: 10.1145/2487575.2488200

[14] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *nature*, vol. 381, no. 6582, p. 520, 1996. doi: 10.1038/381520a0

[15] Census at School Canada, "Average reaction time", http://censusatschool.ca/data-results/2016-2017/average-reaction-time/ , 2017.

[16] Human Benchmark, "Reaction time statistics," https://www.humanbenchmark.com/tests/reactiontime/statistics , 2018.

[17] J. Priebe, "A study of internet users' cookie and javascript settings," Apr. 2009. [Online]. Available: http://www.smorgasbork.com/2009/04/29/a-study-of-internet-users-cookie-and-javascript-settings/

[18] A. Winnicki, "Just how many web users really disable cookies or javascript?" Apr. 2016. [Online]. Available: https://blog.yell.com/2016/04/just-many-web-users-disable-cookies-javascript/

[19] P. O'Sullivan, E. H. Stern, R. C. Weir, and B. E. Willner, "Pixel cluster transit monitoring for detecting click fraud," Feb. 2016, US Patent 9,251,522.

[20] B. E. Willner, E. H. Stern, P. J. O'Sullivan, R. C. Weir, and S. Callanan, "Cursor path vector analysis for detecting click fraud," Jan. 2015, US Patent 8,938,395.

[21] A. Abraham, "Rule-based expert systems," *Handbook of measuring system design*, 2005. doi: 10.1002/0471497398.mm422

[22] K. Dave and V. Varma, "Predicting the click-through rate for rare/new ads," *Center for Search and Information Extraction Lab International Institute of Information Technology Hyderabad, INDIA*, 2010. doi: 10.1145/1835449.1835671

[23] B. Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson, "What's clicking what? Techniques and innovations of today's clickbots," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011, pp. 164–183. doi: 978-3-642-22424-9_10

[24] J. Parsons, "The difference between website impressions and clicks," Jan. 2015. [Online]. Available: https://growtraffic.com/blog/2015/01/difference-website-impressions-clicks

[25] A. Juels, S. Stamm, and M. Jakobsson, "Combating click fraud via premium clicks." in *USENIX Security Symposium*, 2007, pp. 17–26.

[26] S. Clifford, "Microsoft sues three in click-fraud scheme," Jun. 2009. [Online]. Available: http://www.nytimes.com/2009/06/16/business/media/16adco.html

Paulo S. Almeida is from Brasília, Brazil. There, he obtained a bachelor's degree in computer engineering at UnB (University of Brasília) in 2017.

He worked as an intern for Autotrac Comércio Telecomunicações S/A for one and a half years. He is currently working as a Researcher for Laboratory LATITUDE in University of Brasília. Mr. Almeida is interested in research on the areas of security and networks.

João J. C. Gondim is from Recife, Brazil, where he obtained a degree in electrical engineering (electronics) at UFPE (Pernambuco Federal University) in 1984.

He was also awarded an MSc in Computing Science at Imperial College, University of London, in 1987 and a PhD in Electrical Engineering at UnB (University of Brasília, 2017). Since 1994, Professor Gondim is a lecturer at Department of Computing Science (CIC) at UnB where he is a tenured member of faculty. His research interests are information and cyber security. João is married and has one daughter and two sons. He likes hiking and off roading on a 1966 willys jeep; cooking and wines; assembling robots and watching the stars with his kids.