A TRADEOFF-BASED SECURITY MODEL AGAINST CLICK SPAM ORIGINATED BY SINGLE IP ADDRESSES

N.Zingirian, M.Benini

Department of Information Engineering - University of Padova - ITALY

ABSTRACT

This paper shows a vulnerability of the pay-per-click accounting of Google Ads to the attacks of a malicious single agent and proposes a statistical tradeoff-based approach to reduce this vulnerability. The contribution of this paper is a model to calculate the overhead cost per click necessary to protect the subscribers from click spam and a simple algorithm to implement this protection.

KEYWORDS

Pay-per-click Advertising, Google Ads, Web Advertising

1. INTRODUCTION

The pay-per-click accounting method adopted by Google Ads service for online advertising services [1][2] enables the Advertising Provider (AdP, e.g., Google) to automatically charge the Advertising Subscribers (AdS) for each single advertised page access, i.e., the "click", activated by each web user. Differently from previous online pay-per-click methods, this method does not need AdP and AdS to agree on respective web access logs and "referrer headers" [3], as a consequence the management cost of subscriber's signup and charging processes are kept extremely low. This method allows a single AdP to manage millions pay-per-click contracts, thus making the pay-per-click advertising a mass service, as it appears today.

Considering that the click count is the key number upon which such a mass business calculates huge turnovers, the paper presents a contribution to the very relevant topic of assessing the accuracy of such a number. In particular, the paper evaluates the robustness against malicious web agents, who might be motivated to spam the click counts, for instance, to attack a target AdS to exhaust its daily budget, eventually making it disappear from the advertising network during the first minutes of each accounted day [4].

While big effort has been spent to setup heuristics to detect distributed click spam [5][6][7] [8], this paper shows (Section 2) how a malicious single user agent (web client program), bound to one IP address only, can make the click count increase for a given AdS, even if the charged clicks do not correspond to any real advertisement.

The paper contributions are

- a statistical trade-off model that shows how to make the accounting scheme much safer for the AdS versus the AdP's risk of losing a small percentage of revenues (Section 3),
- a simple algorithm to take advantage operatively of the trade-off model (Section 4), and

David C. Wyld et al. (Eds) : CNSA, DMDB, AIAP - 2018
pp. 79– 86, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80106

- a simulation that validates the model (Section 5).

A discussion of the results and possible extensions concludes the paper (Section 6).

2. ATTACK

The attack presented in this paper is directed to the "fairness principle" of the AdP, that is to charge one click only to the AdS, even when an user agent accesses the same advertised web page more than once. This principle considers that only the first click corresponds to a real advertisement, whilst the other subsequent clicks, referred to as "repeated clicks" in this paper, do not provide any benefit in terms of advertised contents.

The vulnerability to this attack depends on the fact that no deterministic and secure rules to implement this principle are applicable on the AdP server side, to determine whether two clicks are originated by the same user agent or not.

According to our experiments, the approach followed by Google algorithm adopts the heuristic to consider *n* clicks as originated by the same user agents according two conditions

(a) if the user agent "cookie" header values is the same in the *n* HTTP requests corresponding to the *n* clicks OR

(b) if the *n* clicks are originated by the same IP address and also *n* DNS queries are originated from the same IP address to resolve the AP Server name, just before each of the *n* HTTP request corresponding to the clicks,

otherwise the *n* clicks are both accounted, as if they were originated by two different user agents.

```
HTTP_header  = "
      Host: <attacked server>
      Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
      ;q=0.8
      Upgrade-Insecure-Requests: 1
      Accept-Language:it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4,es;q=0.2,pl;q=0.2
      Connection: keep-alive
      User-Agent: //is set with a random choice of user-agent
      Accept-Encoding: */* "

repeat N times // N is number of repeated clicks
      clean_DNS_cache();
      //No Cookies
      HTTP_Send("GET www.google.it/search?q=<QUERY> HTTP/1.1"+HTTP_Header);
      HTTP_Response = HTTP_receive()
      Response_body = HTTP_parse(ENTITY_BODY, HTTP_Response)
      Target_link = search(TARGET, Response_body) // Attack Target
      MyCookie = http_parse(COOKIES, HTTP_Response)
      HTTP_send ("GET target_link HTTP/1.1" +HTTP_Header+"Cookie =" + MyCookie)
      HTTP_Response = HTTP_receive()
      Status_Code = http_parse(STATUS_CODE,HTTP_Response)
      while (Status_Code == 300) // Redirect
            Location = http_parse (LOCATION,HTTP_Response)
            HTTP_Send "GET Location HTTP/1.1" + HTTP_Header + "Cookie = MyCookie"
            Extract Status_Code from HTTP_Response
      end while
end repeat
```

Figure 1 - Pseudo code of the attack

Figure 1 - Click count report before the attacks, 11 clicks per day.

Consequently, the attack works as follows. A malicious HTTP client, whose pseudocode is available in Figure 1, performs many HTTP requests to the same link advertised by the AdP server, simply resetting both the cookie header value and the client's system DNS cache before sending each HTTP request. The client sends many requests from the same source IP address. In case of attack, our AdP has detected only about 42% of false clicks, while the others are normally accounted. As shown in Figure 1 the clicks accounted when attacks take place are much more than the regular average clicks per day shown in Figure 2.

Figure 2 - Click count report after the attacks, only 43 false click detected over 73

Figure 2 and 3 shows the clicks accounted by Google Ads, before and after our attack respectively.

The reader could wonder why Google heuristic does not consider the IP address as a safe information to determine if the same user agent originates two clicks. The answer is that two or even more hosts, each running a different user agent, might send distinct HTTP requests from the same IP address to the same AdP server, as most IP networks over the Internet are IPV4 network adopting the Network Address Translation. The AdP server might see, in that case, different clicks of really different users coming from the same IP address.

We observe that the experiment shows that the heuristic adopted by Google Ads is clearly safe for the AdP but it is unfortunately unsecure for the AdS.

Our investigation explores how to use the probability, for any network, that two independent clicks, coming from the same IP address, originate from two different user agents, to decide whether to count or not to count the clicks.

3. TRADE-OFF MODEL

The model analyzes the clicks coming from a NAT address space using the following variables

- Time interval *T*, called memory interval, within which two clicks are counted only once if they are activated by the same user and have the same target. If the time distance between two following clicks is more than time *T*, then they are counted twice even when directed by the same user to the same advertised resource.
- Integer number *A*, corresponding to the cardinality of the NAT address pool.
- Integer number *C* corresponding to the number of clicks coming from any address in the NAT pool and directed to the same AS resource

The average number *N(A,C)* of "repeated clicks", is the numbers of clicks directed to the same resource and coming from the same IP address but from different users over a NAT pool consisting of *A* addresses, provided that *C* clicks in total are coming from that NAT pool.

N(A,C) is calculated considering that each user performs only one click, so that two different users using the same IP address originate repeated clicks, excluding any click spam event.

We define as loss factor *L*

$$L(A,C) = N(A,C) \times A / C$$

the average percentage of real clicks that are potentially lost (i.e. not accounted) if all multiple clicks (from the second one on), coming from the same address are systematically ignored. The average number of clicks N exceeding the single click is multiplied by the number A of addresses available, to obtain the average click over all the NAT network, and is divided by C to calculate the percentage over all clicks. This percentage corresponds to the loss rate of revenue that is paid by the AdP to protect the AdS.

If *L* is below a fixed threshold, e.g., 1% the protection heuristic decides to ignore all repeated click.

To calculate N(A,C) we consider the clicks as uniformly random independent events falling in a 1-d continuous space that is splitted in A equal segments, each representing the subspace of probability that a click is originated by the address represented by that segment. We assume that there is no correlation between IP address and user interest for a specific content, as there is no reason to suppose any correlation.

This corresponds a Poisson Distribution where $\lambda = C/A$.

Considering that the average number of repeated events falling in the same interval is

$$\sum_{c=2}^{C} (c-1) \cdot P(clicks = c) = \sum_{c=2}^{C} (c-1) \cdot \frac{\lambda^c e^{-\lambda}}{c!}$$

Then, as shown in the Appendix,

$$N(A,C) = \sum_{c=2}^{C} (c-1) \cdot \frac{(C/A)^c e^{-C/A}}{c!} = \frac{C}{A} + e^{-C/A} - 1$$

it is worth noticing that, as shown in the Appendix, if A >> C then

$$N(A,C) < \frac{1}{2} C^2/A^2$$

As a consequence

$$L(A,C) < \frac{1}{2} C \quad /A$$

This approximation, being a second-order Taylor polynomial approximation, is very precise for typical C/A values for large networks, e.g. If C/A = 10^{-3} then $\left|L(A,C) - \frac{1}{2} C/A\right| < 10^{-6}$

4. ALGORITHM

The proposed algorithm bases on the following entities:

1. A table, called the *status table* having the following fields:
   - dest : the destination URL of the Click
   - source: the source IP address (possibly NATted)
   - net: the id of the smallest network range registered in whois DB
   - time: timestamp (epoch)

2. An object, called click, having the properties of each click received, i.e.,
   - click.dest : the destination URL of the Click
   - click.source: the source IP address (possibly NATted)
   - click.time: the click message receiving timestamp (epoch)

Figure 4 shows the pseudo code of a click handler. The actions done after each click is to count the click as valid incrementing the click counter through the function `increment_counter()` or to reject the click through the function `discard` .The algorithm discards the clicks as long as the average statistical number of repeated clicks is below a given threshold.

```
click_handler (click) {
        NET = lookup_net_by_ip(click.source); // from whois DB
        A = lookup_net_size(NET); // smallest whois DB net range
        delete status_table where time < click.time - T; //removes oldest history
        // calculates C
        C = select count(*) from status_table where dest = click.dest and net = NET;
        if (select count(*) from status_table where dest = click.dest and source = click.source >
0)
        { // if more than one click from that source.
        if (0.5 * C / A < threshold )  {
                    discard(click);
                    return; //Exits the click handler
                    }
        }
        increment_counter(click.dest); // click is accounted for invoicing
        insert (click.source, click.dest, NET, click.time) into status_table;
}
```

Figure 4 Pseudo code of trade-off based protection algorithm

5. SIMULATION

We considered in our simulation the Italian provider Vodafone IT having 28.870.000 subscribers and 5.538.048 IP addresses registered. Supposing an advertisement having a huge impact e.g., the percentage of 0.1% over the whole population clicks the same advertised link then C= 28.870 and $C/A = 5.21 \cdot 10^{-3}$.
According to the model presented in Section 4, the click loss ratio is ½ $C/A = 2.6 \cdot 10^{-3}$ if repeated clicks for each IP address are never accounted i.e. it is less than 0.26 %.

A simulation has been carried out to confirm the model.

The simulation programs:

- distributes N users randomly over the all IP address
- selects randomly N/1000 users who clicks the advertised link
- Counts how many clicks have been originated by each IP address
- Yields the number of IP addresses that originated 2 or more clicks divided by the total number of addresses

The simulation has executed 100.000 times and yielded the average value of 2.5368×10^{-3}.

The difference between model and simulation is:

$$|2.6065 \times 10^{-3} - 2.5368 \times 10^{-3}| = 6{,}9652 \times 10^{-5}$$

As a result, the impact of the loss is very low, even in case in which the number of clicks *C* is very high, and the discrepancy between the model and the simulation is negligible.

6. CONCLUSION

The tradeoff protection model presented in this paper allows the AdP precisely assigning a part of investment as an insurance to protect the customers from this very simple attack. The threshold

used in the algorithm exactly corresponds to the percentage of turnover loss that can be decided by the AdP. Accepting this loss, the AdP protects the customer from click spam coming from a single IP address.

This model is particularly suitable for the large mass of small AdS who receive a few clicks per day for which even a moderate repeated attack could completely vanishes their investments. In that case, the customer could get the option of paying an additional security fee corresponding to e.g., 0.5% to 2% of the click price to get the protection against this attack, deciding the algorithm threshold accordingly. Alternatively, the AdP can set the threshold to modulate internal security costs to maintain appropriate service levels.

The time interval *T* should be large enough to collect enough statistics on *C*, and small enough to keep the memory of the clicks not too large to count the accesses of the same users who click again the same resource after long time.

This latter aspect is the key of evolution of the presented algorithm, as the standard deviation of the repeated clicks will be also considered to calculate the loss more precisely, possibly postponing the calculation when the mean is considered enough significant.

7. APPENDIX

By replacing $\lambda = C/A$, N(A,C) can be rewritten as

$$N(\lambda) = \sum_{c=2}^{C} (c-1)\frac{\lambda^c e^{-\lambda}}{c!}$$

and can be splitted into two parts:

$$N(\lambda) = \sum_{c=2}^{C} c\frac{\lambda^c e^{-\lambda}}{c!} - \sum_{c=2}^{C} 1 \cdot \frac{\lambda^c e^{-\lambda}}{c!}$$

The first part yields:

$$\sum_{c=2}^{C} c\frac{\lambda^c e^{-\lambda}}{c!} = \sum_{c=0}^{C} c\frac{\lambda^c e^{-\lambda}}{c!} - \lambda e^{-\lambda} - 0 =$$
$$= \lambda - \lambda e^{-\lambda} - 1$$

because $\sum_{c=0}^{C} c\frac{\lambda^c e^{-\lambda}}{c!}$ corresponds exactly to the Poisson mean λ

The second part yields

$$\sum_{c=2}^{C} \frac{\lambda^c e^{-\lambda}}{c!} = \sum_{c=0}^{C} \frac{\lambda^c e^{-\lambda}}{c!} - (e^{-\lambda} + \lambda e^{-\lambda})$$
$$= 1 - (e^{-\lambda} + \lambda e^{-\lambda})$$

because $\sum_{c=0}^{C} \frac{\lambda^c e^{-\lambda}}{c!} = 1$ as it is exactly the sum of the whole distribution.

Composing the two parts, we obtain

$$N(\lambda) = \lambda - \lambda e^{-\lambda} - 1 - (e^{-\lambda} + \lambda e^{-\lambda}) =$$
$$= \lambda + e^{-\lambda} - 1$$

Expanding N(λ) the Taylor's series of f(x) up to the second order calculated in 0 we obtain:

$$N(\lambda) = N(0) + N'(0)(\lambda - 0) + \frac{N''(0)(\lambda-0)^2}{2!} + R_2(\lambda)$$

$$= (1-1) + (1-1)\lambda + \frac{1}{2}\lambda^2 + R_2(\lambda) =$$

$$= \frac{1}{2}\lambda^2 + R_2(\lambda)$$

Lagrange theorem states that there exists $\xi \in (0,\lambda)$ such that $R_2(\lambda) = -1/6\, e^{-\xi}\lambda$. As a consequence $R_2(\lambda) < 0$.

$$N(\lambda) = \tfrac{1}{2}(\lambda)^2 + R_2(\lambda) < \tfrac{1}{2}(\lambda)^2$$

Replacing λ with C/A we obtain:

$$N(C/A) = \tfrac{1}{2}(C/A)^2 + R_2(C/A) < \tfrac{1}{2}(C/A)^2$$

REFERENCES

[1] Q. Zhang, T. Ristenpart, S. Savage, and G. M. Voelker. Got traffic?: an evaluation of click traffic providers. In Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality, WebQuality '11, 2011.

[2] Yuan, Shuai, Ahmad Zainal Abidin, Marc Sloan, and Jun Wang. "Internet advertising: An interplay among advertisers, online publishers, ad exchanges and web users." arXiv preprint arXiv:1206.1754 (2012).

[3] T. Berners-Lee, J. Gettys, R. Fielding, J. Mogul, L. Masinter, P. Leach and H. Frystyck, Hypertext Transfer Protocol-HTTP/1.1, RFC 2616, HTTP Working Group, June 1999.

[4] V. Dave, S. Guha, and Y. Zhang. Measuring and Fingerprinting Click-Spam in Ad Networks. In ACM SIGCOMM Conference on Data Communication, 2012

[5] L. Zhang and Y. Guan. Detecting Click Fraud in Pay-Per-Click Streams of Online Advertising Networks. In Proceedings of the IEEE Conference on Distributed Computing Systems, 2008

[6] Exposing Click-Fraud Using a Burst Detection Algorithm, D. Antoniou, M. Paschou, E. Sakkopoulos, E. Sourla, G. Tzimas, A. Tsakalidis, E. Viennas

[7] M. Taneja, K. Garg, A. Purwar, S. Sharma, Prediction of click frauds in mobile advertising Contemporary Computing (IC3), 2015 Eighth International Conference on, IEEE (2015)

[8] H. Haddadi. Fighting Online Click-Fraud Using Bluff Ads. SIGCOMM CCR, 40(2):22–25, Apr. 2010.