ELSEVIER

TORONTO '99

On the security of pay-per-click and other Web advertising schemes

Vinod Anupam *,a,1, Alain Mayer a,1, Kobbi Nissim b,2, Benny Pinkas b,2,3,
Michael K. Reiter a,4

[a] *Bell Laboratories, Lucent Technologies, Murray Hill, NJ, USA*
[b] *Department of Computer Science and Applied Math, Weizmann Institute of Science, Rehovot, Israel*

Abstract

We present a hit inflation attack on pay-per-click Web advertising schemes. Our attack is virtually impossible for the program provider to detect conclusively, regardless of whether the provider is a third-party 'ad network' or the target of the click itself. If practiced widely, this attack could accelerate a move away from pay-per-click programs and toward programs in which referrers are paid only if the referred user subsequently makes a purchase (pay-per-sale) or engages in other substantial activity at the target site (pay-per-lead). We also briefly discuss the lack of auditability inherent in these schemes. © 1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: Electronic commerce; Secure systems; On-line advertising; Pay-per-click

1. Introduction

Click-through payment programs ('pay-per-click') have become a popular branch of Internet advertising. In the simplest case, the Webmaster of the site running the program, here called the target site, agrees to pay each referrer site for each user who clicks through the referrer to the target. That is, if a user views a Web page served from the referrer site, and then clicks on a hypertext link (e.g., banner ad, logo) in that page to the target site, then the target site owes the referrer site some predetermined amount of money. The target site runs a click-through payment program in order to motivate the referrer to prominently display ads for the target site on its Web pages. Often, the target site does not administer such a program itself but rather employs a third-party ad network to administer the click-through program on its behalf[5].

Click-through counts are also used by the Internet advertising industry at large to determine the effectiveness of a banner ad (its location, design, etc.). Often the click-through rate (i.e., the percentage of users who clicked on the banner after seeing it) is used as a metric to determine the cost of placing the banner ad on a particular Web page[6].

As has been recognized in the click-through pay-

[1] Corresponding author.
[1] E-mail: {anupam, alain}@research.bell-labs.com
[2] E-mail: {kobbi, bennyp}@wisdom.weizmann.ac.il
[3] Research partly supported by an Eshkol fellowship from the Israeli Ministry of Science.
[4] E-mail: reiter@research.bell-labs.com

[5] Examples of such third-party services include ClickTrade (http://clicktrade.linkexchange.com/), eAds (http://www.eads.com/), and ValueClick (http://www.valueclick.com/).
[6] See BannerNetwork (http://adnetwork.linkexchange.com) and BannerSwap (http://www.bannerswap.com/).

ment industry, click-through payment programs are susceptible to *hit inflation*, where a referrer artificially inflates the click-through count for which it should be paid. Consequently, most ad networks include clauses in their service agreement that explicitly prohibit hit inflation by the referrer and mention that they have "effective software to detect such misuse".

The goal of this paper is to explore the extent to which hit inflation can be detected or prevented in click-through payment programs. The main result of this paper is negative: we present a hit inflation attack that on one hand is very difficult for the target site (or the ad network site, if present) to detect conclusively and that on the other hand can be used by the perpetrating referrer to inflate its referral count at the target site. The attack allows the referrer to transform every visit by a user on any site that is collaborating with the referrer into a click through to the target. We have tested the attack with both Netscape Navigator and Microsoft Internet Explorer browsers.

The practical implications of our attack are potentially significant. If our attack becomes commonplace, then it could accelerate a move away from pay-per-click programs and toward advertising programs where payment is offered to a referrer only if the referred user either makes a purchase at the target site (*pay-per-sale*) or shows some demonstrable interest (*pay-per-lead*). Such variations of click-through programs have already appeared on the Web, presumably motivated by the desire of target sites to pay only for 'high quality' referrals. Our attack is ineffective against pay-per-sale and pay-per-lead programs. However, as we will discuss, these programs are susceptible to another form of fraud that present Web infrastructure offers little ability to detect.

Aside from its potential impact, our attack employs an interesting technique. In the attack, two collaborating Web sites 'team up' so that whenever a user visits one of these sites, the click-through count of the other Web site is incremented at the target. Moreover, this is invisible to the user, and the target has little ability to detect that this is not a legitimate referral, even if its Webmaster suspects that the attack is happening. Rather, to convincingly detect this attack, the Webmaster of the target must locate the Web page on the site that is initiating the attack (i.e., the one that the user actually visited), which should be very difficult unless the target has prior knowledge of the collaborating Web sites.

The rest of this paper is organized as follows. We introduce the hit inflation problem in more detail in Section 2. We describe our attack in Section 3, and we discuss the security of alternative advertising schemes (pay-per-sale and pay-per-click) in Section 4.

2. The hit inflation problem

In order to understand the hit inflation problem, we first must understand how a legitimate click-through is manifested in HTTP protocol messages. Our initial treatment is for the simple case of a click-through program run directly by a target site for referrers. The case of a third-party click-through program provider will be discussed subsequently.

Let R denote a referring site, T denote the target site, and U denote a user's Web browser. A click-through begins when U retrieves a Web page `pageR.html` from R that contains a hypertext link to a page `pageT.html` on site T (see Fig. 1). When the user clicks on that link, the user's browser issues a request to site T for `pageT.html`. An important component of this request is the `Referer` header of the HTTP request for `pageT.html`. This header is set by the user's browser and names the Web page that 'referred' the user to `pageT.html`, in this case `pageR.html`. T uses this `Referer` header to

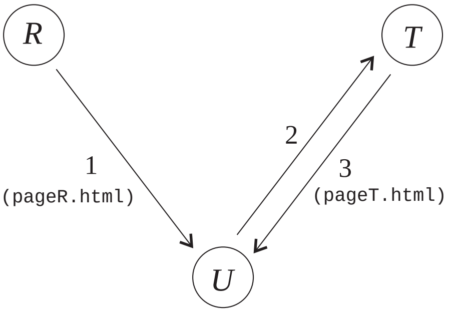


Fig. 1. A click-through: User U retrieves `pageR.html` from R (message 1) and clicks on a link in it, causing `pageT.html` on site T to be requested (message 2) and loaded (message 3).

record the URL of the page that referred the user to `pageT.html`, along with the IP address of *U*. *T* then returns `pageT.html` to *U* for display in the browser.

In a click-through payment program, *T* will periodically pay *R* some previously agreed-upon amount for each click-through from *R* to *T*. The fact that *T* pays for click-throughs provides to *R* an incentive to mount hit inflation attacks on *T*, in which *R* somehow causes *T*'s record of click-throughs from *R* to be increased above the correct number. Here we do not define precisely what the 'correct number' is. Rather, we simply characterize a hit inflation attack as one in which *T* receives a request for `pageT.html` with a `Referer` header naming `pageR.html` when no corresponding Web user clicked to `pageT.html` after viewing `pageR.html`. For example, a straightforward attempt to inflate *R*'s click-through count is for the Webmaster of *R* to run a program that repeatedly sends requests of the appropriate form to *T*. However, because most click-through programs pay only for 'unique' referrals, i.e., click-throughs from users with different IP addresses, multiple click-throughs where the user is at the same site are counted as only one click-through for payment purposes. On the side we remark that counting unique IP addresses is becoming increasingly ineffective, as more user requests are directed through proxy servers either due to the default configuration of the user's ISP (e.g., 99% of AOL subscribers) or to enhance user privacy[7].

A sophisticated attacker could issue multiple requests to *T* with forged IP source addresses, thereby circumventing the unique referrals rule. However, this requires a further degree of technical sophistication and effort on the attacker's part (see, e.g., [3]). Moreover, these attacks can be detected by *T*, due to the fact that in all likelihood, no browser will receive the response from *T*. So, for example, if `pageT.html` is constructed with links to images or other HTML elements that a browser would immediately retrieve upon interpreting `pageT.html`, then a request for `pageT.html` with a forged IP source address will not be followed by requests for the HTML elements contained in `pageT.html`. If it is feared that the attacker will go one step further and even issue these follow-up requests in a predictive fashion to avoid detection, then *T* can dynamically generate `pageT.html` each time with links to different URLs (in the limit, containing a nonce in the URL), thereby foiling any such attempt by the attacker to predict the URLs to request. The end result is that requests with forged IP addresses will stand out to *T* as those for which correct follow-up requests were not received. Moreover, the perpetrator of this attack will be revealed by the `Referer` field of these requests, as this `Referer` field must indicate the referrer that is trying to inflate its hits.

Because of the difficulty and detectability of IP address forgery attacks, probably the most common form of hit inflation today is one in which the referrer *R* forces the user to visit the target *T* by constructing `pageR.html` so as to automatically 'click' the user to `pageT.html` (e.g., see [6]). This simulated click can be accomplished using constructs that will also play a role in our attacks; we thus defer an explanation of these techniques to Section 3. This simulated click can be visible to the user, in which case the user will see, e.g., a new window popped up on his screen unsolicited and containing `pageT.html`. Alternatively, the window can be hidden from the user (e.g., behind the window containing `pageR.html`), so that the user is unaware that she is being 'used' by *R* to gain payment from *T*. Regardless of whether this hit inflation is visible to the user, it is still the case that these attacks can be detected by *T* if the Webmaster of *T* periodically visits the Web pages of the referrers that she pays (preferably from a machine outside her own domain, to avoid detection by the referrer). By inspecting the constructions in those Web pages, and observing the behavior of these pages when interpreted by her browser, the Webmaster of *T* can detect that hit inflation is occurring. Indeed, this examination could even be automated, as it suffices to detect if the referrer's page, when interpreted, causes a request to *T*'s site automatically.

There are numerous variations on click-through programs as described above. In particular, in a program run by a third-party provider, the interaction differs from the above description in that the third party takes the place of *T*. The third party records the click-through and then redirects the request to the actual target site. Another variation is that some click-

[7] Example privacy-enhancing proxies include the Anonymizer (http://www.anonymizer.com) and the Lucent Personalized Web Assistant (http://lpwa.com).

through programs do not make use of the HTTP Referer header, but rather simply have each referrer refer to a different URL on the target site. This approach has the advantage of not relying on the Referer field to be set correctly and thus functioning in conjunction with privacy-enhancing tools that eliminate the Referer field in the HTTP header. However, this approach exposes the click-through program to additional risks: in particular, the referrer Webmaster can broadcast-email (‘spam’) his own banner ad to increase its click-through count. Thus, most click-through programs of this form explicitly prohibit spamming to increase click-throughs, and will cancel the referrer’s account if the referrer is detected doing so.

None of these variations deter the attack we present in Section 3. On the contrary, if the Referer header is not used by the target site, then our attack becomes easier, as will be discussed in Section 3.

3. Undetectable hit inflation for click-through counts

In this section we describe an approach to hit inflation that is very effective on two counts: it enables a referrer to inflate hits arbitrarily, and it does so in a way that is very difficult for the target to detect, even if the target suspects that the attack is being conducted. The attack is equally applicable to both direct click-throughs from a referrer to a target and third-party click-through program providers. Here we present our attack in the context of a direct click-through program. Its full implications will be discussed in Section 3.3.

In our attack, the referrer site *R* inflates its click-through count by translating hits on *another* site *S* that it controls into referrals from site *R* to the target site *T*. That is, when a user visits a certain pageS.html on site *S* — which may have no apparent relationship with site *R* — this has the side effect of causing a click-through to be credited to pageR.html at site *T*. The Webmaster of site *T* can detect this only if she happens to stumble upon pageS.html and examines it carefully. However, if she has no reason to suspect a relationship between *R* and *S*, then confirming this attack is effectively as difficult as exhaustively searching all pages on all Web sites to find pageS.html, i.e., the page that is originating the hit inflation attack. In particular, retrieving pageR.html for examination is of no assistance to the Webmaster of site *T* in detecting this attack.

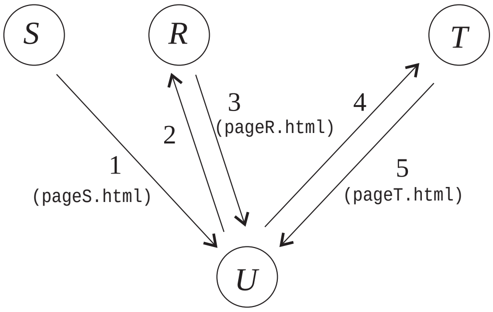


Fig. 2. The attack: User *U* retrieves pageS.html from *S* (message 1), which simulates a click to pageR.html (message 2). In response to this request (with a Referer field of pageS.html), *R* returns a manipulated version of pageR.html (message 3) that simulates a click to pageT.html (message 4). *T* receives a request for pageT.html with a Referer field of pageR.html, causing *R* to be credited for the click-through. For any request that *R* receives that does not name pageS.html as its Referer, *R* returns the benign version of pageR.html.

At a very high level, the attack works as follows; see Fig. 2. The page pageS.html causes a ‘simulated click’ to pageR.html on site *R*. As mentioned previously, this simulated click can be done in a way that is invisible to the user. This simulated click will cause the user’s browser to send a request to site *R* with a Referer field naming pageS.html on site *S*. In response to this request referred by site *S*, site *R* returns a *modified version* of pageR.html to the browser that in turn causes a simulated click to pageT.html, the target page. This causes the browser to request pageT.html from *T* with a Referer field naming pageR.html, thereby causing *T* to credit site *R* with the referral. However, in response to any request for pageR.html that does not contain a Referer field naming pageS.html, site *R* returns the normal and innocuous pageR.html that, in particular, does not simulate a click to pageT.html. So, if the Webmaster of site *T* retrieves pageR.html herself, the page she retrieves yields no evidence of foul play. In the following subsections, we detail the components of this attack.

3.1. Simulated clicks

A component of our attack is the ‘simulated click’, in which one Web page (the referrer) causes the user’s browser to request another Web page (the target) on another Web site, with a `Referer` field naming the referrer. Indeed, our attack of Fig. 2 consists essentially of two simulated clicks, one from *S* to *R* and one from *R* to *T*. The preservation of the `Referer` field is critical for a simulated click (and our attack), and this requirement rules out some of the most straightforward possible implementations: e.g., if the referrer serves a page that ‘refreshes’ the browser to the target page using HTML’s `<meta>` tag (see [4]), then this retrieves the target page but does not preserve the `Referer` field. As discussed in Section 1, simulated clicks are already practiced in hit inflation attacks on the Web today. However, presently there seems to be little attempt to hide these simulated clicks from users (e.g., see [6]), whereas we use techniques to hide simulated clicks from users to limit detectability of our attack and the annoyance caused to users.

One feature that makes simulated clicks possible is that modern browsers transmit `Referer` information not only for pages requested by explicit user clicks, but also for components embedded in pages like images, and especially subdocuments like frames and layers (see, e.g., [4] for an introduction to these constructs in HTML). For example, the Web page containing a layer is named in the `Referer` header of the request for the document contained in the layer, even though no user clicks are involved when the layer contents are retrieved. Therefore, a simple and effective simulated click can be achieved for Netscape Navigator 4.x (NN4) and Microsoft Internet Explorer 4.x (IE4) if the referring site serves a page with a layer that contains the target page (NN3 and IE3 do not support layers). To hide this simulated click from the user, the layer containing the target page can be made of zero size, or stacked below the containing document so that it is invisible to the user. Another form of simulated click can be achieved using frames with IE3 and IE4, since these browsers report the document containing a frameset as the `Referer` for the documents in each of the frames. Thus, a referrer can create an invisible, simulated click to a target by serving a page that contains a frameset with a zero-size frame that loads the target page. Interestingly, NN3 and NN4 report the `Referer` of the page containing the frameset as the `Referer` for each of the documents in the frames. Thus, we use layers to conduct a subdocument-based simulated click in NN4. It is somewhat more awkward to perform a subdocument-based simulated click in NN3. In order to use the appropriate form of simulated click, the server can determine the user’s browser and version from the `User-Agent` header in the browser’s request.

For reasons that will be described in Section 3.2, these subdocument-based forms of simulated click will not suffice to make our attack as effective as it can be. Rather we will also employ JavaScript for explicitly simulating a click on a link (see, e.g., [2] for more information about JavaScript). When a JavaScript script in a Web page causes this simulated click on one of its own links, the browser behaves as if the user clicked on the link, and thus requests the URL of the link and loads it into the browser. In order to hide this simulated click from the user, the referring page can cause the contents of the ‘clicked’ URL to be loaded into a separate window that lies beneath the user’s browser window. Then the referring page can quickly close this window once the referred-to page has started loading, or after a brief duration in which the page should have started loading. The attentive user might notice an additional window indicated on her desktop toolbar for a brief moment, but otherwise this additional window will almost certainly go unnoticed by the random user for the brief period of time in which it is present. And even if the user does notice the additional window, the JavaScript script can still prevent the user from exposing it before it is closed by repeatedly raising the main browser window above it.

The JavaScript mechanism to simulate a click on a link differs slightly from browser to browser, and care must be taken to ensure that this simulated click preserves the `Referer` field received by the target. In IE4, link objects support the `click()` method that, when invoked, causes the browser to behave as if a user clicked on the link. Referrer information is preserved, i.e., the document containing the link is reported to the target Web site as the `Referer`. In NN3 and NN4, as well as in IE3, link objects do not have the convenient `click()` method. However, using a script to send the browser window to the URL corresponding to the link causes the script's page to be reported as the referrer to the target Web site.

To summarize, the attack that we detail in the following section will use two different forms of simulated clicking. The first employs a subdocument (i.e., layer or frame) form of simulated click in the referring page and will be called a *subdocument-based* simulated click. The second employs JavaScript and will be called the *JavaScript* simulated click.

3.2. Detailing the attack

As described in Section 3.1, at a high level our attack consists of two simulated clicks, one from *S* to *R* and one from *R* to *T* (see Fig. 2). However, the nature of these two simulated clicks is quite different. Recall that *S* and *R* are collaborating in this attack, and indeed it is important for the attack that in the first simulated click, *R* recognizes that the simulated click from *S* is happening (so that it can serve the 'attack' version of `pageR.html` that causes the simulated click to *T*). On the other hand, in order to make our attack truly undetectable to *T*, it is important that *T* be *unable* to detect that the referral from *R* is by a simulated click. Because of these conflicting requirements, the two simulated clicks in our attack are conducted via different mechanisms.

The simulated click from *S* to *R*, so that *R* recognizes the simulated click from *S*, is the easiest to achieve. Since *S* and *R* are in collaboration, their Webmasters can set up the Web sites so that *any* request that *R* receives for `pageR.html` with a `Referer` field of `pageS.html` is by a simulated click from *S*. This can be ensured if `pageS.html` has no link to `pageR.html` that can be clicked by the user. Thus, the subdocument-based approach of Section 3.1, in which the only link to `pageR.html` is for a layer's contents, for example, is ideally suited for this simulated click.

The simulated click from *R* to *T* is more sensitive, as it is essential that *T* be unable to detect that the click is simulated. In particular, if JavaScript is enabled in the browser, then a script in `pageT.html` could detect the subdocument-based simulated click of Section 3.1. Specifically, in current browsers `pageT.html` can use JavaScript to detect whether it is displayed in a frame. Moreover, in version 4 browsers, `pageT.html` can use JavaScript to detect the size of its window, layer, or frame, and thus `pageT.html` can be designed to detect the case when it is displayed in a zero-size frame or layer. For these reasons, `pageR.html` must test for various conditions when conducting its simulated click and tailor its method of attack to them. Specifically, the simulated click from *R* to *T* should occur as follows:

(1) `pageR.html` first tests if JavaScript is enabled in the browser. If not (i.e., JavaScript is disabled), then it simulates a click to `pageT.html` using the subdocument-based simulated click of Section 3.1.

(2) If JavaScript is enabled in the browser (and thus `pageT.html` has greater detection capabilities at its disposal), then `pageR.html` performs the simulated click using the JavaScript method of Section 3.1 that directs `pageT.html` to a new window, hidden from the user.

There is always the possibility that the Webmaster of site *T* will request `pageR.html` for inspection, and so we remind the reader that for any request for `pageR.html` that does not name `pageS.html` as the `Referer`, *R* should respond with an innocuous Web page that does not simulate a click to *T*.

3.3. Discussion

The attack detailed in this section is effective even if a third-party click-through program provider is used. In this case, *T* is the third-party provider and not the actual target site, but this distinction has no bearing on the mechanism behind our attack. Another difference is that third-party programs often do not make use of the `Referer` header for identifying the referrer, but rather simply use a different URL per referrer. In this case, however, our attack just becomes easier since there may be less of a need to retain the correct `Referer` header when performing simulated clicks.

Our attack has other implications. As mentioned in Section 2, most click-through programs are not agreeable to the use of spamming by a referrer to increase click-through counts, and in fact, many click-through programs explicitly prohibit the use of spamming in their contracts with referrers. Our attack, however, makes target sites susceptible to 'indirect' spamming that is hard to detect: a spammer (an agent of S) can drive a large number of users to S, triggering the inflation attack. The lack of an obvious relationship between R and the spammer or S makes it difficult for the Webmaster of T to detect this practice.

Many click-through programs desire 'high quality' referrals, i.e., referrer sites with a targeted audience (e.g., technology oriented sites). Our attack enables a referrer site R with appropriate content to register in the click-through program, while using a different site S with completely different content to attract the click-throughs. Furthermore, many click-through programs disallow referrers with illicit material, regardless of their popularity. Our attack enables referrers R to use such sites to draw users and register click-throughs for R at the target.

To see the potential for profit from this attack, consider that the average click-through rate for banner ads is 1–3%, and that payments for click-throughs are calculated accordingly. Our attack can yield an effective rate of almost 100% for users who visit `pageS.html` and thus (unknowingly) click through `pageR.html` to `pageT.html`. We can go a step further and use S in conjunction with several (say 10) sites $R_1, \ldots, R_{10}$ that are enrolled in different click-through programs, and thereby get an effective click-through rate of 1000%. This is undetectable as long as the different target sites do not compare the IP addresses from which they receive clicks at the same time. (Thus, this multi-target attack might be impossible with target sites that are on the same third-party click-through program).

An attacker might draw suspicion if the target site T monitors the click through rate (CTR) of its ads. The target can monitor the CTR if R's page is required to load the ads from a site that is controlled by the target. A high click-through rate (say greater than 5%) is likely attract the attention of the target's Webmaster, if only to learn the marketing practices of the referrer. The attacker can prevent such inquiries by keeping the CTR low. One way to achieve this is to register site R with, say, 20 different targets. Whenever R receives a request with a `Referer` field naming `pageS.html`, it returns a page containing ads for each of the targets, and performs a simulated click on one of these ads at random. The attacker is paid for 100% of the visits to S, while keeping the CTR below 5% at each target. This method can of course be extended to achieve lower CTR or higher payment rates.

Another way for the target site T to detect the attack is to search for pages that have links to `pageR.html`, in an effort to find `pageS.html`. A simple approach would be to use existing search engines to find pages that refer to `pageR.html`[8]. However, S can easily avoid detection by serving a different, benign version of `pageS.html` to spiders of search engines[9]. A second approach that T can try is to perform the search for pages like `pageS.html` itself, using a spider. This reconnaissance operation is of almost the same scale as building a search engine, and can be complex and costly. Moreover, R and S can extend the attack in a natural way to use a chain of three or more simulated clicks, e.g., from some S' to S to R to T. This further complicates efforts to 'trace backward' along the chain to find the page that initiates the attack.

Probably the most viable way of detecting the attack is for T to monitor user activity (e.g., mouse movement, mouse clicks, or filling out a form) on T's site. A real user will typically either click further into the site or leave the site immediately. The former is easily detectable and confirms the existence of a real user. To detect the latter case, `pageT.html` could be constructed to include a 'Back' button that both returns the user to the referrer page and informs T that the user clicked on this button. However, this does not capture the case that a user next directs her browser to a bookmarked location, uses the browser's 'Back' button to leave T's site, or closes the window containing `pageT.html`. Similarly, `pageT.html` could be constructed with JavaScript code to inform T of mouse movement over `pageT.html`, or to inform T of the length of time that the page was active in the browser (e.g., by causing a message to be sent to T every few seconds). The latter offers little information to T if `pageR.html` closes the window containing `pageT.html` after a random amount of time.

[8] For example, InfoSeek (http://www.infoseek.com) explicitly offers the ability to query for such information, via a 'field search' using the syntax link:pageR.

[9] Search engine spider queries can be easily identified as such. For example, see http://searchenginewatch.com/webmasters/spiders.html.

The former, i.e., detecting mouse movement over `pageT.html`, possibly lets *T* confirm that a user sees the page (if the user moves the mouse over it). However, again it does not enable *T* to determine that a user did not see the page. In the limit, *T* could occasionally serve a version of `pageT.html` that contains a newly generated question for the user to answer (and perhaps offers a financial incentive to do so), to see if a user responds.

While none of these techniques can offer proof that the attack is taking place, they can offer *T* statistical evidence of the attack if the attack is mounted aggressively through a single referrer *R*. As such, detecting user activity seems to be the most promising direction for coping with this attack, and in fact is the same principle that is behind pay-per-lead and pay-per-sale schemes discussed in Section 4.

Finally, it is worth noting that legal means could also be used to discourage hit inflation attacks. Extreme hit inflation attacks could be grounds for a civil lawsuit if detected. If the threat of civil action is combined with suitable criminal penalties, these threats may effectively deter large-scale hit inflation.

4. Pay-per-sale and pay-per-lead

If pay-per-click programs are going to be deemphasized in the future, then it is worth considering the security of the programs that are likely to replace them. Presently, the foremost alternative to pay-per-click programs are programs in which target sites pay only for 'high quality' referrals, i.e., for referred users who perform some substantial activity or make purchases at the target site. There are essentially two forms of such programs:

- *Pay-per-lead:* Referrers are paid only if the user has performed a significant action at the target site, e.g., if she registered an account at the target site or performed successive hits at the target site for more than five minutes.
- *Pay-per-sale:* Referrers are paid some commission for purchases the user makes at the target site [10]. Typically the referrer displays a link for a specific item for sale at the target site, and is paid some percentage for purchases of this item by referred users.

Payments in these programs are typically larger than in pay-per-click programs, since they are more valuable for the target sites.

It is virtually impossible for referrer sites to mount useful hit inflation attacks on such schemes, since simple clicks are worthless to the referrer. However, these programs are susceptible to a different form of fraud, known as *hit shaving*. In hit shaving, the target site fails to report that a referred user executed a lead or sale, thereby denying the user's referrer rightful payment (regardless of whether a third-party program provider is used). Current Web technology offers referrers little ability to detect such fraud (cf. [5]), short of the Webmaster of a referrer site simply clicking through her own site to the target and, e.g., making a purchase to verify that her site is credited with this referral. This type of detection can be powerful: even if the target site attempts to shave just 5% of the commissions it is required to pay, this fraud is expected to be discovered after only 20 such probes by the referrer. However, this type of detection is not always feasible, for example if the target site sells rather expensive items (e.g., cars). In such cases, the referrers are presently at the mercy of target sites to faithfully report the leads and sales for which the referrers should be paid.

Future versions of browsers may provide a mechanism to enable a referrer to monitor the user's behavior at a target site to a limited extent. Specifically, at Bell Labs we have designed and implemented a new JavaScript security model in the free Mozilla source code (see [1]), and this is presently being considered for inclusion in Netscape 5.x browsers. This security model enables (among other things) a JavaScript program from one domain to interact with a page loaded from a different domain, provided that the latter page allows this. This feature can be used to enable the referrer site to monitor the user's actions at the target site with a JavaScript program. For example, this program could report to the referrer when the user registers or makes a transaction at the target site. The model further allows the target site to have fine-grained control over what information on a page is made available to the referrer; e.g., on the sales page, the target could make the total purchase

[10] An example is the Associates program run by Amazon (http://www.amazon.com).

amount available without revealing what books the user bought or the user's credit card information.

Even this new security architecture, however, does not provide machinery sufficient to fully address the hit shaving problem in pay-per-lead and pay-per-sale programs. This is the case for two reasons. First, the additional exposure of user activities to referrers that is enabled by this security architecture, which seems to be needed to combat hit shaving, may be an unacceptable privacy intrusion for many users. And consequently, the security architecture of [1] allows this exposure only with user consent. Thus, the Web advertising industry may need to consider ways to motivate users to allow greater exposure of their Web activities to referrers, in order to combat the threat of hit shaving. Second, a more common and unintentional form of hit shaving occurs when a user clicks from a referrer to a target, exits the browser, and then returns directly to the target later to explore the site or make a purchase. In this case, the referral is (perhaps unintentionally) 'shaved', and foreseeable Web infrastructure offers little machinery for the referrer to detect this.

5. Conclusion

Pay-per-click programs are a popular form of advertising incentive on the Web today. We have presented a hit inflation attack on these programs that appears to be virtually undetectable to target sites and very effective in inflating referral counts. Our attack involves two collaborating Web sites, where each user's visit to the first causes a target to register a referral from the second. There seem to be no sure ways of detecting this attack, short of locating the page on the first site that initiates the attack, though testing by the target site to attempt to determine if a user sees its page may give some indication to the target.

In our opinion, this attack brings the viability of pay-per-click programs into question and, if practiced widely, may accelerate an ongoing trend to move toward pay-per-sale and pay-per-lead programs. As discussed in Section 4, these programs have fraud problems of their own that seem difficult to address given today's Web infrastructure. How to achieve sufficient auditability to eliminate fraud in these Web advertising schemes remains an open problem.

Acknowledgements

We are very grateful to the anonymous reviewers for the 8th International World Wide Web Conference, who offered useful comments on how to improve the presentation of this work.

References

[1] V. Anupam and A. Mayer, Secure Web scripting, IEEE Internet Computing 2 (6) (1998).

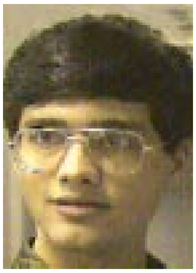
[2] D. Flanagan, JavaScript: The Definitive Guide, 3rd ed., O'Reilly and Associates, 1998.

[3] R.T. Morris, A weakness in the 4.2 BSD Unix TCP/IP software, Computer Science Technical Report 117, AT&T Bell Laboratories, February 25, 1985.

[4] C. Musciano and B. Kennedy, HTML: The Definitive Guide, 2nd ed., O'Reilly & Associates, 1997.

[5] M.K. Reiter, V. Anupam and A. Mayer, Detecting hit shaving in click-through payment schemes, in: Proc. 3rd USENIX Workshop on Electronic Commerce, September 1998, pp. 155–166.

[6] Web surfers wary of 'kidnapping' sites, USA Today, September 28, 1998, http://www.usatoday.com/life/cyber/tech/ctd540.htm.



Vinod Anupam is a member of the Database Systems Research Department in the Systems and Software Research Center of Bell Labs, Lucent Technologies. He received a Ph.D. in computer science from Purdue University in 1994. His research interests include collaborative computing, Internet and Web security, electronic commerce, graphics and visualization, and mobile computing.

Alain Mayer is a Research Scientist in the Secure Systems Research Department at Bell Labs/Lucent Technologies. He joined Bell Labs in September 1996 from System Management Arts (SMARTS), a network management start-up company. He received his Ph.D. in computer science in 1995 from Columbia University. Alain's research interests include electronic commerce, network and Web security, cryptography, privacy, and network management. During 1999, he is serving on the program committee of both the USENIX Security Symposium and the ACM Conference on Computer and Communications Security.

Benny Pinkas is a Computer Science doctoral student at the Department of Applied Math and Computer Science of the Weizmann Institute of Science, Rehovot, Israel. He received his B.A. (Summa Cum Laude) and his M.Sc., both in Computer Science, from the Technion — Israel Institute of Technology, in 1988 and 1991, respectively. During 1991-1996 he served in the Israel Defense Forces, where he worked in computer science and communications research and development. His main research interests are computer security and cryptography, and in particular communication efficient security protocols. His research is supported by an Eshkol Fellowship from the Israeli Ministry of Science.

Michael K. Reiter (www.bell-labs.com/user/reiter) is Department Head of the Secure Systems Research Department in Bell Laboratories, Lucent Technologies. He received the B.S. degree in mathematical sciences from the University of North Carolina in 1989, and the M.S. and Ph.D. degrees in computer science from Cornell University in 1991 and 1993, respectively. During 1998–2000, he will serve as Program Chair of the flagship computer security conferences of both the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE). Dr. Reiter's research interests include all areas of computer and communications security, electronic commerce, and distributed computing.