Badvertisements: Stealthy Click-Fraud with Unwitting Accessories

Mona Gandhi Markus Jakobsson Jacob Ratkiewicz
Indiana University at Bloomington

Abstract

We describe a new type of threat to the Internet infrastructure, in the shape of a highly efficient but very well camouflaged click-fraud attack on the advertising infrastructure. The attack, which we refer to as a "badvertisement", is described and experimentally verified on several prominent advertisement schemes. This stealthy attack can be thought of as a threatening mutation of spam and phishing attacks, with which it has many commonalities, except for the fact that it is not the targeted individual who is the victim in the attack, but the unwitting advertiser.

Keywords: advertisements, click-fraud, dual-personality page, JavaScript, phishing, spidering, stealth.

1 Introduction

An increasing portion of advertising revenue is drawn from Internet advertising. Advertising programs such as AdBrite [1], Miva AdRevenue Express [2], Banner Box [3] and Clicksor [4] generate a fair portion of the per-click revenue for publishers on the Internet. We present a click-fraud attack that is applicable to these advertisement schemes – and very possibly others as well[1]. As indicated by a series of lawsuits [5, 6, 7, 8], most of the recent click fraud attacks are based on traditional malware. Our approach, on the other hand, demonstrates a new attack vector that increases the threat by not hinging on any client-side vulnerability. This is quite worrying, given how much easier it is for a fraudster to make an accessory visit a webpage than it is to infect a machine with malware.

The attack allows a fraudster to force unwitting accessories to perform *automated click-throughs* on ads hosted by the fraudster, resulting in revenue generation for the fraudster at the expense of advertisers. In spite of appearing to be clicked on by the accessories, the advertisements are not visible to them, which reduces the risk of complaints. Being in direct conflict with the terms of service on two counts, our attack covers its tracks to prevent discovery performed using *reverse spidering*, which is the commonly favored technique to verify compliance with the terms of service. We refer to this new type of offense as a *badvertisement*.

Just as phishing can be thought as having sprung out of the con artist's attempts to sell the Brooklyn bridge to passersby, our attack can be thought of as an Internet-powered version of a common trick used by street corner "ad artists" that – paid per delivered pamphlet – hand out two ads per recipient instead of one. However, the parallel to phishing serves as more than an analogy of the evolution of attacks. As many of the components of these two types of attacks are common, we believe that there is a substantial risk that our techniques can become heavily deployed as a revenue alternative to collecting personally identifiable information. Thus, the high degree of commonality between the attack delivery vectors can turn this attack into an overnight disaster [9, 10]. While current click-fraud is mostly malware based – at least as suggested in by recent legal proceedings against Overture [5] – we believe that our attacks demonstrates that this may soon change, and that the problem may still be in its infancy.

Our attack works by artificially – and stealthily – increasing click-through counts of ads hosted by a fraudster or his associates. This is accomplished (and was experimentally verified[2]) by corrupting the JavaScript file that is required to be downloaded and executed by a client's web browser to publish sponsored advertisements. A part of this JavaScript code traverses the source of the advertisement frame to determine a list of legitimate ads and then simulates an artificial click. Each click in effect increases the number of visits registered for a sponsored ad and thereby accounts for higher per-ad cut revenue for the publisher. While it may at first appear that this attack should be easily detected by inspection of the click-through rates from the domain in question, this is unfortunately not so. This is the case since the fraudster can cause *both* click-throughs and non-click-throughs (in any desired proportion) to be generated by the traffic accessing the corrupted page, while keeping end-users corresponding to both of these classes unaware of the advertisement. We note that a fraudster can even generate click-fraud revenue from traffic to a site that is not allowed to display advertisement from a given provider – a typical example of such a site would be a porn site. We show how such abuse can be disguised by the fraudster. It is also worth noting that owners of sites that are used to generate revenue for a "badvertiser" may be unaware of the attack they are part of. Namely, given the invisibility of the attack, domain owners can remain unaware of the existence of an attack mounted by a corrupt webmaster of theirs, or a person able to impersonate the same. The latter ties the problem back into phishing once again.

A second contribution of this paper is a description of countermeasures to our attack. In addition to describing countermeasures based solely on clever construction of advertisement code, we also consider abuse detection schemes that take place after the fact – to detect click fraud, in the case that preventing it is not possible. These schemes fall into two basic categories – *active* schemes that attempt to seek out instances of click fraud, and *passive* schemes that watch for click fraud in progress.

Definitions for technical terms commonly used in our paper can be found in the Appendix.

2 Related Work

The attacks described in this paper consist of two components: *delivery* and *execution*. The first component either brings *users to the corrupt information*, or *corrupt information to the users*. The second component – which we focus on herein – causes the automated but invisible display of an advertisement to a targeted user.

Considering the delivery component, bringing users to the corrupt information may either rely on sites that users visit voluntarily and intentionally due to their content, or on sites that may contain information of no particular value, but which users are tricked to visit. Users may be tricked to visit sites using known techniques for page rank manipulation (see, e.g., [11]).

Fraudsters may also use spam as a delivery approach, and try to entice recipients to click on a link, causing a visit to a site with badvertisements. Sophisticated spam-driven attempts may rely on user-specific context to improve the probabilities of visits. For example, a fraudster may send out email that appears to be an electronic postcard sent by a friend of the recipient; when the recipient clicks on the link, he will activate the badvertisement. An 80% visit rate was achieved in recent experiments by Jagatic et al [12] by using relational knowledge collected from social network sites.

Bringing corrupt information to users may simply mean sending emails with content causing an advertisement to be clicked; this is possible whenever the targeted user has a mail reader in which JavaScript is enabled. In such situations, successful execution will be achieved exactly when the fraudster can cause the spam email in question to be delivered and viewed by the targeted user. It is worth noting that recent developments among spammers frustrate content-based filtering by means of programmatic obfuscation techniques. These techniques typically employ text mixed with noise data in the body of the email message, combined with a programmatic means (such as Cascading Style Sheet code) of separating the two when the message is viewed. Theoretically speaking, it is clear (following a reduction to the halting theorem) that there are obfuscation techniques that cannot be de-obfuscated without execution of the corresponding script. Such emulation is not currently undertaken, both due to its potential cost implications (in terms of execution time), and due to the lack of a strong motivation. It is worth noting that the delivery component bears strong resemblance to delivery techniques used in phishing, but it is also much more general and therefore harder to protect against.

Our attack relies on clients having JavaScript enabled. However, this is not a real limitation, both since 90% of web browsers have it enabled[3] and because the advertising services studied in this paper themselves count on users having JavaScript enabled. Thus, the execution component of the attack relies simply on a JavaScript trick that causes an ad to be automatically clicked and processed by a client's web browser. This causes the click to be counted by the server side, which in turn triggers a transfer of funds from the advertiser to the domain hosting the advertisement, i.e., the fraudster.

The camouflage techniques a fraudster could deploy to avoid detection during reverse spidering are simple instances of a stateful black-box adversary that behaves differently when probed than when regularly used. Other – and more complex – cases of such behavior can be found among other places in the realm of SETUP attacks [13], where hardware or software will behave correctly until triggered by some event, just to avoid detection of the behavioral modification of the code during testing.

Note that our key approach is to display effectively different text to the user than to the policy verification bot. This is not as simple as serving a different page to anybody identifying itself as a bot; it is, for example, known that Google performs some amount of spidering without identifying itself as Google. This is how it was discovered that BMW Germany attempted to raise its own PageRank by serving a keyword-laden page to the GoogleBot only [14].

Click fraud has been acknowledged as a major threat against the current generation of ad services that sell clicks to advertisers; thus, necessarily, many efforts have arisen to combat it. Among these are techniques which attempt to identify fraudulent clicks. Some of these identify fraudulent clicks by simply watching for a large number of clicks to originate from a single IP address. This simplistic attack would bear no resemblance to one mounted by the techniques we describe. Other fraud detection techniques, such as [15], work by statistically learning average click through rates for ads and then detecting deviations from that norm. Again, it is doubtful that even these techniques could detect the abuse we describe, as this abuse doesn't rely on huge numbers of clicks over short periods of time, but rather a slight, across-the-board increase on clicks on all ads on a site.

(a)

```
function print_ads() {
  for( each ad ) {
    document.write( text of ad );
  }
}
```

(b)

```
(function () {
  function print_ads() {
    document.write( "<iframe src=url of ad server>" );
  }

  print_ads();
})()
```

Figure 1: Insecure ad display strategy compared with unattackable (AdSense) strategy. Pseudocode for the general strategy used by most providers to print ads is given in (a) – note that each ad is inserted directly in the current document. AdSense's more secure strategy is shown (in greatly simplified form) in (b). Here, an anonymous function wraps all declarations to keep them out of the global namespace. Further, the downloaded script does not print the ads itself; it prints an iframe which draws its source from yet another script. This allows browser policies to protect the contents of the iframe from reading by other scripts in the page (such as badvertisment scripts).

Our attack works on a large variety of platforms, but notably, not on Google AdSense. Google's approach is different in two ways from the vulnerable schemes that we show how to attack. One difference lies in that the code that is downloaded from the Google server to fetch the ads (a script called `show_ads.js`). Whereas the attacked schemes allow these scripts to be accessed by another JavaScript component, this is prevented in AdSense by defining all functions inside a single *anonymous function*. This difference is illustrated in Figure 1. A second distinction is that the AdSense code does not rewrite the current page to include the ads; it rewrites the current page to include an `iframe` tag with its source set to the output of yet another script. It is this latter script that actually generates the ads. Since the source of this generated iframe is a Google server, JavaScripts served by other sites (such as any served by a fraudster) are prevented by a browser from accessing it. This would prevent a fraudster from fetching a list of links for auto-clicking.

3 The Making of a Badvertisement

A successful badvertisement is one which is able to silently generate automatic click-throughs on advertisement banners when users visit the site, but remain undetected by auditing agents of the ad provider. Further, while the ad provider may place restrictions on the type of content that may be shown on a page containing their ads, a fraudster would like to be able to show prohibited content (e.g., pornography) on their pages which may draw higher traffic. To this end, we introduce two concepts – a Façade page and a *dual-personality* page. The Façade page is what a visitor to the badvertisment page will see; it shows them content only, hiding advertisements and auto-clicking. Its purpose is to hide the badvertisements from the users. It works in conjunction with a *dual-personality* page, possibly on another domain. This page may change its personality to a "good" or "evil" version depending on whether it is being viewed by an (oblivious) user, or a suspicious ad provider. Figure 2 gives a graphical representation of these two pages. In the following explanations, we suppose the existence of two websites – `www.verynastyporn.com`, which contains the Façade page, and `www.veryniceflorist.com`, which is registered with the ad provider and contains the dual-personality page.

In the following we give a brief overview of how JavaScript is used in these techniques. We then describe the implementation of both the Façade and the dual-personality page.

3.1 JavaScript for Click Fraud

Typical online advertisement services (such as AdBrite [1] and the others we examine) work by providing webmasters a snippet of JavaScript code to add to their pages. This code is executed by the web browser of a visitor to the site, and downloads ads from the advertiser's server at that time. The ad download triggers a rewrite of the frame in which the JavaScript appears, replacing it with the HTML code necessary to display the ads. When a user clicks an advertisement link, they "click through" the ad provider's server, giving the ad provider the opportunity to bill the client for the click. The user is then taken to the ad client's

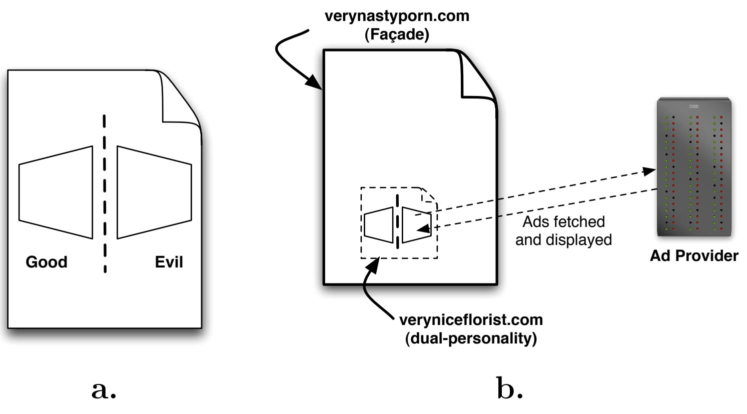


Figure 2: Pages to hide badvertisements from users and from auditors. Part **(a)** shows a dual-personality page; to an oblivious user it is evil, and includes badvertisements (because it will not be detected). However, to a suspicious auditor, it appears good. Part **(b)** depicts one such dual-personality page included in a Façade page, which hides the evil behavior of the dual-personality page from a visiting user.

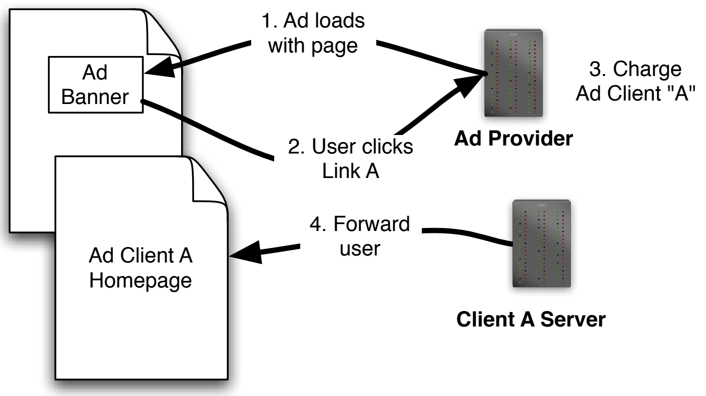


Figure 3: Normal use of the advertising programs we examine. A webmaster who wishes to serve sponsored advertisements places a snippet of JavaScript in his page which downloads ads from the ad provider when the page is loaded. If a user clicks a link for one of those ads, they are forwarded to the ad client; on the way, the click is registered so that the ad provider can bill the advertiser, and pay the webmaster a share.

homepage. Figure 3 illustrates this scenario.

Badvertisements, then, contain additional JavaScript that simulates automatic clicking. This is accomplished as follows: after the advertiser's JavaScript code runs and rewrites the page to contain the ads, the badvertisement JavaScript parses the resulting HTML and compiles a list of all hyperlinks. It then modifies the page to contain an HTML *iframe* or inline frame (designed to allow a page to seamlessly draw content from other pages). This iframe is configured to load its content from the advertiser's site, creating the impression

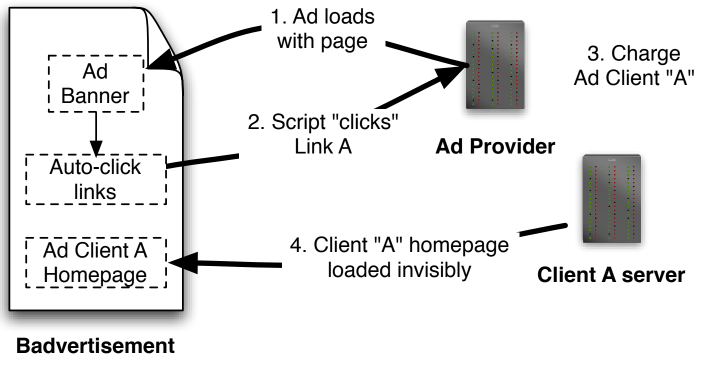


Figure 4: Auto-clicking in a hidden Badvertisment. Compared to Figure 3; the ad banner presented here is hidden. JavaScript code extracts links from the hidden ad banner and causes them to be displayed in an another hidden iframe, creating the impression that the user has clicked these links.

(a)

```
for( i = 1; i <= 10; i++ ) {
   document.write( i );
}
```

(b)

```
code = "f@o"+"#r(i@"+"#="+1+";@"+"i#<=1%0"+";i"
       +"+@#+){@"+"d#oc"+"%um#"+"en%t."+"@w#r"
       +"i#te"+"(@i + \"<b" + "r>\");}";
eval(code.replace(/[@#%]/g, ""));
```

Figure 5: Example of how `eval` may be used to obfuscate JavaScript code. Both statements are functionally equivalent — each prints the numbers 1..10 when executed by a JavaScript interpreter.

that the user has decided to click that link (Figure 4).

It should be noted that JavaScript is used both by the legitimate advertiser and by the badvertiser. The latter uses JavaScript to parse and edit the JavaScript of the legitimate advertiser. As an aside, note that it is possible for JavaScripts to be obfuscated to the point that their effect cannot be determined without actually executing them. This allows the fraudster to hide the fact that an attack is taking place from bots that search for JavaScripts containing particular patterns that are known to be associated with badvertisements. One such obfuscation method is use of the `eval` keyword, which allows JavaScript code to programmatically evaluate arbitrarily-complex JavaScript code represented as string constants. Figure 5 gives an example of how this might be used. In essence, this technique therefore allows the inclusion of arbitrary noise in the script, to make searching for tell-tale signs of click fraud fruitless.

3.2 Hiding the Attack From Clients

As mentioned in the previous section, iframes can be used to create an impression of a click for the advertisement URLs identified using JavaScript. Repeating this process several times will create the impression on the server-side that the user has clicked on several of the advertisement links, even though she may not have. The behavior described here would be implemented in a dual-personality page placed on `veryniceflorist.com`. This entire site would then be included in a zero-sized iframe hidden in the Façade page. Thus, it will invisibly generate clicks that appear to come from `veryniceflorist.com` whenever a user visits `verynastyporn.com`.

3.3 Hiding the Attack From Ad Providers

Concealing the badvertisement from the legitimate ad provider is somewhat more difficult than hiding it from users. The auditing method commonly used by ad providers is a web spider that follows the referrer links on clicked advertisements – that is, when an ad on a page is clicked, an auditing spider may choose to follow the referrer link to the page that served the clicked ad. The goal of the fraudster must then be to detect when the page is being viewed by an auditing spider, and to serve a harmless page for that instance. The fraudster does this with the dual-personality page; the entire process is as follows (also summarized in Figure 6):

1. Create a personalization CGI script which runs when a user visits the Façade. This script simply assigns a visitor a unique ID and redirects them to a second CGI script, passing the ID as a `GET` parameter so that it appears in the URL.

2. This second CGI script outputs the Façade, complete with its script placeholder. This placeholder is configured to run a third CGI script, also passing the ID along.

3. This third CGI script may now choose which JavaScript to generate. If the ID passed has never been visited before, the CGI script will generate an "evil" JavaScript; if the ID has been previously visited, the CGI script will generate some "good" JavaScript. (This script might also use criteria other than an ID to determine which personality to show; for instance, referrer, browser type, or even time of day.)

Considering item 3 in more detail – the "good" JavaScript should be a script that has essentially no effect. The fraudster would probably not wish to serve a blank script as this might look suspicious, so he could instead serve a script that performs some innocuous function. What this function is does not matter – the only reason for its inclusion would be to avoid including an empty JavaScript. The "evil" JavaScript, however, can re-write the page arbitrarily. This rewrite, of course, is simply to include a zero-size iframe that loads `veryniceflorist.com` invisibly, and directs `veryniceflorist.com` to use its evil personality.

The result of all this is that the user, when visiting a site with badvertisements, will see only the original content of the site (without the badvertisements). The advertiser will be given the impression that their ads (as served by the ad provider) are very successful (as they are viewed often). However, the advertiser is of course not getting its money's worth, as the ads are not really being viewed by users.

As an example, consider the sequence of events that will occur when a user visits `verynastyporn.com` for the first time:

1. The visitor will first be served the Façade page, which will invisibly include `www.veryniceflorist.com`. The user will also be assigned a unique ID, which will be passed to `veryniceflorist.com`.

2. A CGI script at `veryniceflorist.com` will output a completely legitimate page (including visible advertisements, and no autoclicking), but will also include a placeholder for another JavaScript.

3. The visitor's browser will request the JavaScript, passing the unique ID. The server will check to see if the ID has already been registered; since it will not be, the server will return the badvertisement JavaScript, and register the ID as visited.

4. The user's browser will interpret the JavaScript, causing the ads in the (invisible) `veryniceflorist.com` site to be auto-clicked.

5. To the ad provider, the clicks will appear to come from `veryniceflorist.com`.

The fraudster may choose to pass on a large proportion of chances to serve badvertisements, in order to be more stealthy. This would be accomplished by modifying step 1 above to not include `veryniceflorist.com` some proportion of the time, or step 4 to not generate auto-clicks sometimes. The fraudster may also randomly choose which ad will be auto-clicked, from the served list. This will ensure that the same IP address does not generate an excessive number of clicks for a single advertisement, as a different IP will initiate each.

Figure 6 illustrates how the website will change its appearance only when safe to do so; consider its behavior when visited by various agents:

- *Human visitor* (at `verynastyporn.com`): The human visitor will not navigate directly to the page; it will likely be invisibly included in the Façade page. Thus, the user will have an unvisited ID, and auto-clicks may be generated by the fraudster. In addition to considering only whether the ID has been visited or not, many other policies could be used to determine whether to serve the badvertisment. For instance, the evil webmaster might choose to serve only good scripts to users coming from a particular IP range, or visiting at a certain time of the day. Using techniques such as those in [16], a webmaster might even choose to serve a particular script only to those users with certain other sites in their history.

- *Human visitor or standard spider* (at `veryniceflorist.com`): Here, the human visitor (or spider) won't have a unique ID, so `veryniceflorist.com` will show its good personality and not include auto-clicking.

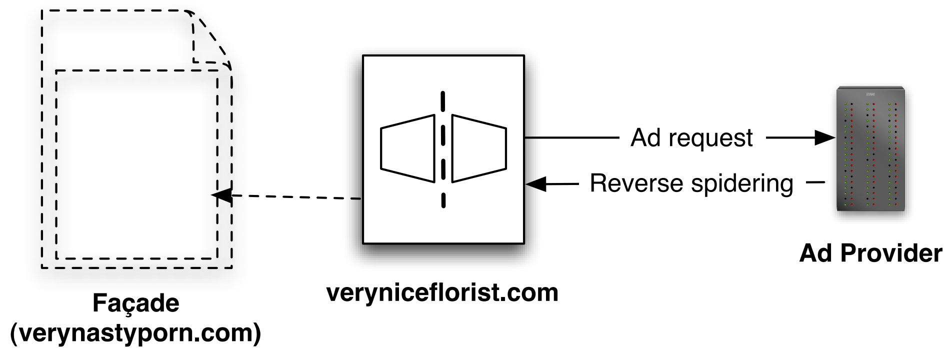


Figure 6: Assignment of IDs prevents any agent not entering through the Façade from seeing the evil JavaScript. The dual-personality page at `veryniceflorist.com` only reveals its evil side when passed an as-yet unvisited ID. When it is given no ID or a visited ID it shows its good side. When included from the Façade page, `veryniceflorist.com` has an unvisited ID and thus contains badvertisments. However, when a user visits `veryniceflorist.com` normally, or when a reverse spider from the ad provider audits it, it will show its good side – this is because these requests contain no ID and a visited ID, respectively.

- *Standard (forward) spider* (at `verynastyporn.com`): Typical web spiders do not evaluate JavaScript in pages. Indeed, it is in general not useful for a web spider to interpret JavaScript, as this is typically used to enhance interaction with users and not to generate text that the web spider might index. Here, however, this fact gives the fraudster an opportunity to hide illicit text from the web spider by causing that text to only become available through the interpretation of JavaScript. Thus, while a spider may visit the Façade page and receive an unvisited unique ID, the fraudster may make the badvertisment page included only after interpretation of JavaScript and thus invisible to the spider.

- *Reverse spider* (at `veryniceflorist.com`): Note that the reverse spider may not enter the site by way of the Façade; it will be following a referrer link on a clicked ad. Any referrer link will contain the unique ID issued by the Façade, and `veryniceflorist.com` will show its good personality to an already-visited ID. Since any ID in a referrer link *must* already be visited (by the user that unwittingly generated the auto-click) the reverse spider will always see the good `veryniceflorist.com` page (even if it evaluates JavaScript).

It should be noted in any case that advertiser's detection of an offending site does not necessarily prevent its owner from earning any illicit revenue — a fraudster might register many `veryniceflorist`-like domains under different assumed names, and parcel traffic over them so that advertiser's detection of any one of them only evaporates a portion of the total profits.

Complex camouflage rules. The criterion for loading the badvertisements can be almost arbitrarily complex (in terms of what the fraudster is able to find out about the user's navigation). This can serve to frustrate detection – as an auditor would have to satisfy the same requirements to see the badvertisements – but decreases the proportion of traffic that the fraudster can monetize. In the long run, however, this actually serves to increase profits, as it delays detection. Here we describe several potential restrictions:

- If two or more sites – say `nastyporn.com` and `verynastyporn.com`, collude, each could cause `veryniceflorist.com` to display invisible badvertisements only to users referred from the other colluding site. That is, `verynastyporn.com` may serve badvertisements only to users referred from `nastyporn.com`, and vice-versa. In general, such "chains" might be arbitrarily long – so badvertisements might only be shown when a user gets to `verynastyporn.com` by way of `nastyporn.com` and `porn.com` in turn. Here, the goal is to expose the badvertisements (and thus run the risk of detection) only when the viewer came from another porn site, and is thus likely a regular user in search of porn (rather than an auditor).

- Badvertisement sites might use various techniques to tell if agents viewing the site are human. These might involve making an agent solve a CAPTCHA [17] or showing badvertisements only to users who have downloaded at least 5 movies or pictures from the site.

- Badvertisements might only be shown to users who have a number of other pornographic web sites in their browser history. This might be determined through the browser probing techniques described in [16, 18, 19].

4 Why will users visit the site?

The previous discussion stated that the fraudster will create a legitimate-appearing site that will be approved by the ad provider, and then create a badvertisement JavaScript function which will be loaded from an offsite server for execution only when the innocent users visit the site. However, there are other ways the webmaster might lure users to visit the site, including the use of mass email and hosting the ads on a pornographic websites (which itself is prohibited). These are detailed below.

Email as a Lure. There can be multiple variations to the medium through which an attack can be mounted. One of them can be through email messages. Email messages can either be spammed indiscriminately to many addresses, or spoofed to users so as to appear to be from their acquaintances (cf. [12]). Each email might contain a "hook" line to encourage users to visit the site, and a link to the portal page. This also reduces the possibility of a scenario of a search engine locating the badvertisement, as the abuser would no longer need to list their portal page on any search engine. Thus, an ad provider must trap such an email in an email honeypot in order to locate the site hosting the portal.

Popular Content as a Lure. Unscrupulous webmasters with popular sites could easily employ this technique to increase their revenue. In particular, sites that host pornography can take advantage of the high traffic these sites typically generate by hosting hidden badvertisements. This can be justified by the results of a small experiment conducted at Berkeley which states that approximately 28% of websites on the Internet contain pornographic content [20]. Although hosting ads with pornography is prohibited by the terms of service by almost all advertising programs, the techniques set forth here could enable a webmaster to hide all illicit content from spiders, showing it only to users. This would be done by re-writing the page on the client side using JavaScript. We consider this type of content distinct from the following type (viral content) in the effort that is required by the webmaster to promote it; here we consider the instance of the owner of an already-popular site simply installing badvertisements behind the scenes.

"Viral" Content as a Lure. A phenomenon of the social use of the Internet is the propagation of so-called "viral content." This content is typically something which is considered interesting or amusing by a recipient, and therefore propagates quickly from person to person [21]. A fraudster could host content likely to promote such distribution on his site (even cloning or stealing it from another site) and then distribute a link to this content through email, hoping that it is passed along. An attack could be thus be perpetrated with minimal bootstrap effort compared to the previous scenario; while the previous scenario assumes an already-established website, here a fraudster need only steal a humorous video from another site (for instance) and spoof emails to a few thousand people, hoping they will be sent along.

5 Detecting and Preventing Abuse

There is an ongoing industry-wide effort to develop tools that will effectively detect and block many common click-fraud attacks. Most of the attacks discovered and reported so far have been malware-based attacks that rely on automated scripts ([22]), individuals hired by competitors [23] or proxy servers used to generate clicks for paid advertisements. Companies like AdWatcher[24] and ClickProtector[25] have initiated efforts to counter this. The essence of their approach is to track IP addresses of machines generating the clicks, as well as identifying the domain from which the clicks are registered. By collecting large logs and performing expert analysis, irregularities such as repeated number of clicks for a certain advertisement from a particular IP address, a particular domain or abnormal spikes in traffic for a specific website are identified. However, as described in sections 3.2 and 3.3, the stealthy attack described in this paper will go undetected by any of these tools. It is therefore of particular importance to determine other unique mechanisms of detecting and preventing attacks of this nature. These can be divided into two classes: *active* and *passive*. Our proposal for the former of these is intended to detect click-fraud attempts housed on webpages that users intentionally navigate to (whether they wanted to go there, or were deceived to think so) where as the proposal for the latter is suited for detection of email-instigated click-fraud.

- An *active* client-side approach interacts with search engines, performs popular searches, and visits the resulting sites. It also spiders through sites in the same manner as users might. To hide its identity, such an agent would not abide the `robots.txt` conventions, and so, would appear as an actual user to the servers it interacts with. The agent would act like a user as closely as possible, including occasionally requesting some advertisements; it would always verify that the number of ad calls that were made correspond to the number of requests that a human user would perceive were made. (The latter is to detect click-fraud attempts in which a large number of ad requests are made after a user initiates a smaller number of actual requests.)

- A *passive* client-side approach observes the actions performed on the machines of the person appearing to perform the click. This may be done by running all JavaScript components in a virtual machine (appearing to be a browser) and trapping the requests for advertisements that are made. Any webpage that causes a call of a type that should only be made after a click occurred can be determined to be fraudulent. While this takes care of the type of automated click-fraud described herein, it would not defend against a version that first causes a long (and potentially random) delay, and then commits click-fraud *unless* the virtual machine allows randomly selected scripts to run for significant amounts of time, hoping to trap a delayed call. We note that excessive delays are not in the best interest of the fraudster, as his target may close the browser window, and therefore interrupt the session before a click was made. Passive client-side solutions may be housed in security toolbars or by anti-virus softwares.

  Another type of *passive* solution is an infrastructure component. This would sift traffic, identify candidate traffic, and emulate the client machine that would have received that packets in question, with the intent of identifying click-fraud. A suitable application might be an ISP-level spam filter or an MTA. Before emails are delivered to recipients, they could be delivered in virtual machine versions of the same, residing on the infrastructure component, but mimicking the client machine.

For all of the above solutions, it should be clear that it is not necessary to trap all abuse. Namely, even if a rather small percentage of abuse is detected, this would betray the locations that house click-fraud with a high likelihood that increases with the number of users that are taken to one and the same fraud-inducing domain. (On the other hand, it is clear that a fraudster would not want to limit the number of victims too much, either, else the profit would become rather marginal.) Evidence from analysis show that if only a small percentage of users have the client-side detection tool installed, this will make attacks almost entirely unprofitable, given reasonable assumptions on the per-domain cost associated with this type of click-fraud. This is discussed in depth in Section 6.

Collecting evidence. Law enforcement can collect evidence in a way that is structurally similar to how a service provider can detect the presence of an attack. If a company reports a website to have launched badvertisements, then law enforcement can collect evidence of this abuse by re-tracing the steps of the user, that lead to the display of the badvertisement, and record the resulting traffic. If this results in the execution of a badvertisement, then this evidence indicates collusion between the facade site and the dual-personality page, the latter which would have performed click-fraud, and the former of which would have aided in the same.

Tools such as Wireshark [26] can be used to watch for anomalous traffic, including attempts to contact third-party sites for badvertisements. This would be done by viewing suspect pages in a browser with Wireshark running in the background. After the session was over, the auditor would examine all the network traffic captured by Wireshark to determine if third-party sites were contacted.

The presence of badvertisements can also be determined from a careful examination of the suspect page's source. When the source of the page itself does not contain badvertisement code (perhaps because the server elected not to include it, based on the camouflaging techniques described above), auditors can still find conclusive evidence of badvertisements by examining the source of server-side software (e.g. after gaining access to it by subpeona).

6 Economic Analysis

This attack can be very profitable, as shown in the following paragraphs, and thus effective countermeasures are essential. Before beginning the discussion, it is necessary to define some quantities that will be used, as follows:

p	:	Probability of discovery per attack instance (a single attempt against a single user, involving a single domain).
s	:	Probability that a user will visit the domain after the attack is performed against them (that is, that the attack will succeed).
v	:	Probability with which the fraudster foregoes automatic clicking and simply shows the ad to the user.
n	:	Number of attack instances that will be made for each domain (that is, number of attempts that will be made to lure a unique user to one of the domains). Assume this is constant across all domains.
b_v	:	Average monetary benefit for the fraudster, per successful attack instance that results in only a view of an banner.
b_c	:	Average monetary benefit for the fraudster, per successful attack instance that results in one or more clicks on ad(s) in a banner.
c	:	Cost for fraudster to attempt to attract a single user to a single domain (per period).

Let us assume, based on a single preliminary proof-of-concept experiment that we performed, that $b_v = \$0.01$ and $b_c = 0.25$ in the following. Further, let's suppose that $c = 0.1/\text{cent}$ — money that the fraudster spends buying updated content for the site, or renting time on a botnet to send out spam messages, for instance. Also let us assume that the payout period for the click-based advertising scheme is one month, so our revenue figures reflect what can be made in this time period. The fraudster will wish to avoid suspicion by having a surprisingly-large number of users who view the banner click a link, so let's suppose $v = 0.95$.

Our model does not account for an element of time – each period's attacks are considered independent from those of the previous period, for reasons of simplicity. That is, our model is of a fraudster doing the following over the course of a month:

- Day 1: Send out n attacks, each with a probability p of being instantly caught (thus invalidating the whole attack). This costs the fraudster $n \cdot c$ (whether he succeeds or not).
- Day 2..30: $n \cdot s$ users visit the fraudster's site. With probability v the fraudster simply shows them an add (perhaps invisibly). With probably $1 - v$ the fraudster causes the user to click one or more ads.
- Day 31: If the fraudster was not caught, he receives a check from the ad provider for $s \cdot n \cdot (v \cdot b_v + (1 - v) \cdot b_c)$. If he was caught, he receives nothing.

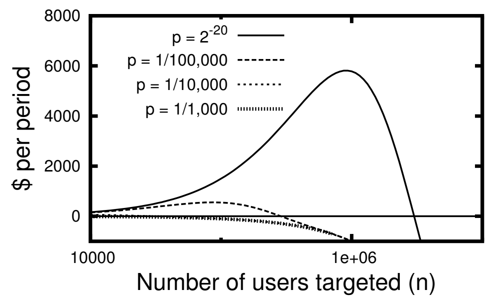
Note that the fraudster can distribute the attack over multiple domains to amortize the risk of being caught (as detection might only mean the loss of profits from a single domain).

More formally, the following general equations then hold:

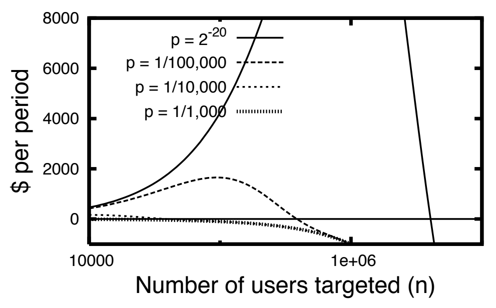
$$\begin{array}{rc} \text{risk of discovery of a domain:} & 1 - (1 - p)^n \\ \text{risk of discovery of all domains:} & 1 - d \cdot (1 - p)^n \\ \text{expected benefit per domain:} & (1 - p)^n \cdot s \cdot n \cdot (v \cdot b_v + (1 - v) \cdot b_c) - n \cdot c \end{array}$$

In the following paragraph, discussion of the economic implications of various countermeasures that might be employed by ad providers is presented. These countermeasures affect only p, the probability that an attack instance will result in the detection of the fraudster. Of course, the fraudster would take care to find a reasonable level of income for a site with a traffic level similar to his, and limit the auto-clicking code in order to remain relatively inconspicuous. Thus, it can be assumed in the following that ad providers cannot use income or frequency as clues to detect fraudulent sites. Further, it is assumed that the fraudster operates from outside the United States, and thus is effectively immune to prosecution from the ad provider; to the fraudster, being caught means nothing more than not being paid (and registering a new account).

No countermeasures. When deploying no countermeasures, the ad provider can find out about the offending site only by either having it reported by someone who stumbles on it, or by having a human at the ad provider visit the site after noticing its unusual popularity. Let us (reasonably) assume that the probability of this happening is very low, and consider "very low" to mean $p = 2^{-20}$. The fraudster's revenue then grows essentially as a linear function of the number of users they choose to target (Figure 7).



(a)



(b)

Figure 7: Benefit for fraudster, in dollars earned, given several probabilities p of detection by ad providers. Note that when $p = 1/1000$, the profit hardly goes above zero; when $p = 1/10000$, profit tops out at around \$500 per domain; after that, the risks quickly outweighs the rewards. Once p shrinks below 1/10000, however, the fraudster fares much better – this is the current threat situation as far as we can tell. Figure (a) shows benefit when the reward per click is \$0.25; figure (b) considers reward per click to be \$1.00.

Active or passive countermeasures Suppose that the variables here combine to yield $p \in \{1/1000, 1/10,000, 1/100,000\}$. The analysis for given hypothetical detection probabilities is shown in Figure 7. Note that even a modest detection probability such as $p = 1/10,000$ limits potential profits considerably.

7 Conclusion

This is a serious attack with significant revenue potential for its perpetrators. Phishers might find attacks of this nature more profitable than identity theft. It is not necessary that these attacks have more revenue potential than phishing attacks, but they are certainly more convenient to perform; the perpetrator can make cash directly, rather than coming into possession of credit card numbers which must be used to buy merchandise to be converted into cash. The execution of this attack does not require significant technical knowledge (assuming the development of a page pre-processor that would insert the malicious code), and thus it could be performed by almost any unscrupulous webmaster.

Further, we believe that this attack would generalize to any system of user-site-oriented advertising similar in operation to AdBrite. This was verified by testing on a few other advertising programs like Miva's AdRevenue Xpress [2], BannerBox [3] and Clicksor [4]. The code proposed to automatically parse and click advertisements can be adapted to parse any advertisements which are textually represented, so long as they can also be parsed by the client browser (and this must be true, of course). Likewise, our techniques for avoiding detection by spiders do not rely on any particular feature of any of these ad providers and would apply equally well to any type of reverse auditing mechanism.

8 Acknowledgments

The authors would like to thank Sid Stamm for technical assistance with the development of the attack. We are also grateful to Jean Camp, Aaron Emigh, Filippo Menczer, and Bennet Yee for helpful feedback on early versions of this paper, and to Fred Cate and Judie Mulholland for helping us locate relevant references. Finally, we are thankful for the very constructive criticism that the anonymous referees provided us with.

Notes

[1]In fact, Google's AdSense [27] is the only advertisement scheme we are aware of that is not susceptible to our attack.

[2]Our attack was only performed a small number of times per affected provider, and with a total revenue generation of less than $5.

[3]This is a commonly quoted estimate, and is based on a January 2005 survey.

References

[1] AdBrite. `www.adbrite.com`.

[2] Miva AdRevenue Express. `www.miva.com`.

[3] Banner Box. `www.bannerbox.co.uk`.

[4] Clicksor. `www.clicksor.com`.

[5] Company Files Fraud Lawsuit Against Yahoo. `www.washingtonpost.com/wp-srv/technology/documents/yahoo_may2006.pdf`, May 1, 2006.

[6] Yahoo settles 'click fraud' lawsuit. `www.msnbc.msn.com/id/13601951/`, June 28, 2006.

[7] Hosting Company AIT Leads Class-Action Suit Against Google. `www.marketwire.com/mw/release_html_b1?release_id=103417`, December 28, 2005.

[8] Click Fraud Gets Day in Court–Maybe. `searchlineinfo.com/Click_fraud_lawsuit/`, April 21, 2005.

[9] Click Fraud Costs Advertisers $800M. `www.internetfinancialnews.com/insiderreports/featured/ifn-2-20060706ClickFraudCostsAdvertisers800M.html`, July 6, 2006.

[10] Are You Clicking Your Money Away. `www.amazines.com/Blogs/article_detail.cfm/137016?articleid=137016&Title=marketing%2Cadvertising%2Cmoney%2Ccash`, July 31, 2006.

[11] Santo Fortunato, Marian Boguna, Alessandro Flammini, and Filippo Menczer. How to make the top ten: Approximating PageRank from in-degree, 2005. *Working paper.*

[12] Tom Jagatic, Nate Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *To appear in CACM*, 2006.

[13] Adam Young and Moti Yung. *Malicious Cryptography.* John Wiley and Sons, 2004. ISBN: 0-7645-4975-8.

[14] German BMW banned from Google. `blog.outer-court.com/archive/2006-02-04-n60.html`.

[15] Nicole Immorlica, Kamal Jain, Mohammad Mahdian, and Kunal Talwar. Click fraud resistant methods for learning click-through rates. *Technical Report*, 2005.

[16] Markus Jakobsson, Tom Jagatic, and Sid Stamm. Phishing for clues. `www.browser-recon.info`.

[17] Luis von Ahn, Manuel Blum, Nicholas Hopper, and John Langford. CAPTCHA: Completely Automated Public Turing Test to Tell Computers and Humans Apart. `www.captcha.net`.

[18] Markus Jakobsson and Sid Stamm. Invasive browser sniffing and countermeasures. In *Proceedings of the 15th International Conference on World Wide Web*, pages 523–532. ACM Press, New York, NY, 2006.

[19] Collin Jackson, Andrew Bortz, Dan Boneh, and John Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th International Conference on World Wide Web*, pages 737–744. ACM Press, New York, NY, 2006.

[20] How much information? `www.sims.berkeley.edu:8000/research/projects/how-much-info-2003/internet.htm`, 2003.

[21] Markus Jakobsson and Sid Stamm. Socially propagated malware. `www.verybigad.com`, 2006.

[22] Feds Arrest Alleged Google Extortionist. `www.internetnews.com/bus-news/article.php/3329281`, March 22, 2004.

[23] India's secret army of online ad "clickers.". `timesofindia.indiatimes.com/articleshow/msid-654822,curpg-1.cms`, May 3, 2004.

[24] AdWatcher. `www.adwatcher.com/`.

[25] ClickProtector. `www.clickprotector.com/`.

[26] Wireshark: A network protocol analyzer. `www.wireshark.org`.

[27] Google AdSense. `www.google.com/adsense`.

Appendix: Terms and Definitions

The following are definitions of terms as used in the context of click fraud.

Dual-Personality Page A page which appears differently when viewed by different agents, or depending on other criteria. Typically one "personality" of the page may be termed "good," and the other "evil."

JavaScript A simple programming language which is interpreted by web browsers. It enables web site designers to embed programs in web pages, making them potentially more interactive. Despite its simplicity, JavaScript is quite powerful.

Phishing Attempting to fraudulently acquire a person's credentials, usually for financial gain.

Referrer When a web browser visits a site, it transmits to the site the URL of the page it was linked from, if any. That is, if a user is at site B and clicks a link to A, when the web browser visits A, it tells A that B is its referrer.

Reverse Spidering When an ad is clicked, the ad provider can track the referrer of the ad as the page that the ad was served from. We use 'reverse spidering' to refer to the ad provider's spidering of this page.

robots.txt File which may be included at the top level of a web site, specifying which pages the webmaster does not wish web spiders to crawl. Compliance is completely voluntary on the part of web spiders, but is considered good etiquette.

SETUP Attack A SETUP attack is one in which a hardware or software provider replaces the expected code with code that performs malicious activities in a stealthy manner, and potentially hidden from inspection by a mechanism that switches from compliant to malicious functionality after a triggering event has occurred.

Spidering Downloading a page by automatic means. Commonly used by search engines in their efforts to index web pages.